IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of	:	THE COMMISSIONER IS AUTHORIZED TO CHARGE ANY DEFICIENCY IN THE FEES FOR THIS PAPER TO DEPOSIT ACCOUNT NO. 23-0975
Hajime OGAWA et al.	:	
Serial No. NEW	:	**Attn: APPLICATION BRANCH**
Filed June 30, 2003	:	Attorney Docket No. 2003_0866A

COMPILER APPARATUS WITH FLEXIBLE OPTIMIZATION

CLAIM OF PRIORITY UNDER 35 USC 119

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

Applicants in the above-entitled application hereby claim the date of priority under the International Convention of Japanese Patent Application No. 2002-195305, filed July 3, 2002, as acknowledged in the Declaration of this application.

A certified copy of said Japanese Patent Application is submitted herewith.

Respectfully submitted,

Hajime OGAWA et al.

By _____
Michael S. Huppert
Registration No. 40,268
Attorney for Applicants

MSH/kjf
Washington, D.C. 20006-1021
Telephone (202) 721-8200
Facsimile (202) 721-8250
June 30, 2003

日 本 国 特 許 庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出願年月日
Date of Application: ２００２年 ７月 ３日

出願番号
Application Number: 特願２００２－１９５３０５

[ST.10/C]: [ＪＰ２００２－１９５３０５]

出願人
Applicant(s): 松下電器産業株式会社

２００３年 ２月２１日

特許庁長官
Commissioner,
Japan Patent Office 太田信一郎

出証番号 出証特２００３－３００９８２１

【書類名】　特許願
【整理番号】　5037740105
【あて先】　特許庁長官殿
【国際特許分類】　G06F　9/45
【発明者】
【住所又は居所】　大阪府門真市大字門真１００６番地　松下電器産業株式会社内
【氏名】　小川　一
【発明者】
【住所又は居所】　大阪府門真市大字門真１００６番地　松下電器産業株式会社内
【氏名】　瓶子　岳人
【発明者】
【住所又は居所】　大阪府門真市大字門真１００６番地　松下電器産業株式会社内
【氏名】　坂田　俊幸
【発明者】
【住所又は居所】　大阪府門真市大字門真１００６番地　松下電器産業株式会社内
【氏名】　高山　秀一
【発明者】
【住所又は居所】　大阪府門真市大字門真１００６番地　松下電器産業株式会社内
【氏名】　道本　昌平
【発明者】
【住所又は居所】　大阪府門真市大字門真１００６番地　松下電器産業株式会社内
【氏名】　濱田　智雄

【発明者】

【住所又は居所】　大阪府門真市大字門真１００６番地　松下電器産業株式会社内

【氏名】　宮地　涼子

【特許出願人】

【識別番号】　000005821

【氏名又は名称】　松下電器産業株式会社

【代理人】

【識別番号】　100109210

【弁理士】

【氏名又は名称】　新居　広守

【電話番号】　06-4806-7530

【手数料の表示】

【予納台帳番号】　049515

【納付金額】　21,000円

【提出物件の目録】

【物件名】　明細書　1

【物件名】　図面　1

【物件名】　要約書　1

【プルーフの要否】　要

【書類名】　明細書

【発明の名称】　コンパイラ装置

【特許請求の範囲】

【請求項１】　ソースプログラムを機械語プログラムに翻訳するコンパイラ装置であって、

生成する機械語プログラムを最適化する旨の指示を取得する指示取得手段と、

取得された指示に従って機械語命令列を生成することにより、最適化を施す最適化手段とを備え、

前記最適化手段は、前記指示取得手段がグローバルメモリ領域に配置する配列データについての指示を取得した場合には、取得した指示に従ってグローバルメモリ領域に配置する配列データを決定することにより、前記最適化を施す

ことを特徴とするコンパイラ装置。

【請求項２】　前記指示取得手段は、前記ソースプログラムを翻訳する旨の指示とともに、グローバルメモリ領域に配置する配列データの最大データサイズについての指定を取得し、

前記最適化手段は、前記ソースプログラムで宣言された配列データのうち、前記最大データサイズを超えないデータサイズの配列データについてはグローバルメモリ領域に配置し、前記最大データサイズを超えるデータサイズの配列データについてはグローバルメモリ領域外のメモリ領域に配置する

ことを特徴とする請求項１記載のコンパイラ装置。

【請求項３】　前記指示取得手段は、前記ソースプログラムの中において、特定の配列データについて、グローバルメモリ領域に配置しない旨の指示を検出し、

前記最適化手段は、前記指示取得手段によって検出された指示の対象となった配列データについては、グローバルメモリ領域外のメモリ領域に配置する

ことを特徴とする請求項１記載のコンパイラ装置。

【請求項４】　前記指示取得手段は、前記ソースプログラムの中において、特定の配列データについて、グローバルメモリ領域に配置する旨の指示を検出し

前記最適化手段は、前記指示取得手段によって検出された指示の対象となった配列データについては、グローバルメモリ領域に配置する

ことを特徴とする請求項１記載のコンパイラ装置。

【請求項５】　高級言語で記述されたソースプログラムが記録されたコンピュータ読み取り可能な記録媒体であって、

前記ソースプログラムには、当該ソースプログラムを機械語プログラムに翻訳するコンパイラに対して、（１）特定の配列データについて、グローバルメモリ領域に配置しない旨を指示する記述、及び、（２）特定の配列データについて、グローバルメモリ領域に配置する旨を指示する記述の少なくとも１つが含まれる

ことを特徴とする記録媒体。

【請求項６】　ソースプログラムを機械語プログラムに翻訳するコンパイラ装置であって、

生成する機械語プログラムを最適化する旨の指示を取得する指示取得手段と、

取得された指示に従って機械語命令列を生成することにより、最適化を施す最適化手段とを備え、

前記最適化手段は、前記指示取得手段がソフトウェアパイプライニングによる最適化についての指示を取得した場合には、取得した指示に従ってソフトウェアパイプライニングに関する最適化を施す

ことを特徴とするコンパイラ装置。

【請求項７】　前記指示取得手段は、前記ソースプログラムを翻訳する旨の指示とともに、ソフトウェアパイプライニングによる最適化をしない旨の指示を取得し、

前記最適化手段は、前記指示取得手段がソフトウェアパイプライニングによる最適化をしない旨の指示を取得した場合に、前記ソースプログラム中の全てのループ処理について、ソフトウェアパイプライニングによる最適化が行われることを抑制する

ことを特徴とする請求項６記載のコンパイラ装置。

【請求項８】　前記指示取得手段は、前記ソースプログラムの中において、特定のループ処理について、ソフトウェアパイプライニングによる最適化をしな

い旨の指示を検出し、

前記最適化手段は、前記指示取得手段によって検出された指示の対象となったループ処理については、ソフトウェアパイプライニングによる最適化が行われることを抑制する

ことを特徴とする請求項６記載のコンパイラ装置。

【請求項９】　前記指示取得手段は、前記ソースプログラムの中において、特定のループ処理について、プロログ部及びエピログ部を除去したソフトウェアパイプライニングによる最適化をする旨の指示を検出し、

前記最適化手段は、前記指示取得手段によって検出された指示の対象となったループ処理については、プロログ部及びエピログ部を除去できる範囲でソフトウェアパイプライニングによる最適化をする

ことを特徴とする請求項６記載のコンパイラ装置。

【請求項１０】　前記指示取得手段は、前記ソースプログラムの中において、特定のループ処理について、プロログ部及びエピログ部を除去しないでソフトウェアパイプライニングによる最適化をする旨の指示を検出し、

前記最適化手段は、前記指示取得手段によって検出された指示の対象となったループ処理については、プロログ部及びエピログ部を除去せずに、可能な範囲でソフトウェアパイプライニングによる最適化をする

ことを特徴とする請求項６記載のコンパイラ装置。

【請求項１１】　前記指示取得手段は、前記ソースプログラムの中において、特定のループ処理の繰り返し回数に関する指定を検出し、

前記最適化手段は、前記指示取得手段によって検出された指示の対象となったループ処理については、指定された繰り返し回数に基づく最適化を行う

ことを特徴とする請求項６記載のコンパイラ装置。

【請求項１２】　前記繰り返し回数に関する指定は、前記ループ処理が繰り返される最低回数であり、

前記最適化手段は、前記最低回数がソフトウェアパイプライニングによって重なり合うイタレーション数以上である場合に、ソフトウェアパイプライニングによる最適化を行う

ことを特徴とする請求項１１記載のコンパイラ装置。

【請求項１３】　高級言語で記述されたソースプログラムが記録されたコンピュータ読み取り可能な記録媒体であって、

前記ソースプログラムには、当該ソースプログラムを機械語プログラムに翻訳するコンパイラに対して、（１）特定のループ処理について、ソフトウェアパイプライニングによる最適化をしない旨を指示する記述、（２）特定のループ処理について、プロログ部及びエピログ部を除去したソフトウェアパイプライニングによる最適化をする旨を指示する記述、及び、（３）特定のループ処理について、プロログ部及びエピログ部を除去しないでソフトウェアパイプライニングによる最適化をする旨を指示する記述の少なくとも１つが含まれる

ことを特徴とする記録媒体。

【請求項１４】　ソースプログラムを機械語プログラムに翻訳するコンパイラ装置であって、

生成する機械語プログラムを最適化する旨の指示を取得する指示取得手段と、

取得された指示に従って機械語命令列を生成することにより、最適化を施す最適化手段とを備え、

前記最適化手段は、前記指示取得手段がループアンローリングによる最適化についての指示を取得した場合には、取得した指示に従ってループアンローリングに関する最適化を施す

ことを特徴とするコンパイラ装置。

【請求項１５】　前記指示取得手段は、前記ソースプログラムを翻訳する旨の指示とともに、ループアンローリングによる最適化をしない旨の指示を取得し、

前記最適化手段は、前記指示取得手段がループアンローリングによる最適化をしない旨の指示を取得した場合に、前記ソースプログラム中の全てのループ処理について、ループアンローリングによる最適化が行われることを抑制する

ことを特徴とする請求項１４記載のコンパイラ装置。

【請求項１６】　前記指示取得手段は、前記ソースプログラムの中において、特定のループ処理について、ループアンローリングによる最適化をする旨の指

示を検出し、

前記最適化手段は、前記指示取得手段によって検出された指示の対象となったループ処理については、ループアンローリングによる最適化を行う

ことを特徴とする請求項１４記載のコンパイラ装置。

【請求項１７】　前記指示取得手段は、前記ソースプログラムの中において、特定のループ処理について、ループアンローリングによる最適化をしない旨の指示を検出し、

前記最適化手段は、前記指示取得手段によって検出された指示の対象となったループ処理については、ループアンローリングが行われることを抑制する

ことを特徴とする請求項１４記載のコンパイラ装置。

【請求項１８】　前記指示取得手段は、前記ソースプログラムの中において、特定のループ処理の繰り返し回数に関する指定を検出し、

前記最適化手段は、前記指示取得手段によって検出された指定の対象となったループ処理については、指定された繰り返し回数に基づく最適化を行う

ことを特徴とする請求項１４記載のコンパイラ装置。

【請求項１９】　前記繰り返し回数に関する指定は、前記ループ処理が繰り返される最低回数であり、

前記最適化手段は、前記最低回数が１以上である場合に、繰り返し回数が０である場合に必要とされるエスケープコードの生成を抑制する

ことを特徴とする請求項１８記載のコンパイラ装置。

【請求項２０】　前記繰り返し回数に関する指定は、前記ループ処理が繰り返される最低回数であり、

前記最適化手段は、前記最低回数がループアンローリングによる展開数以上である場合に、ループアンローリングによる最適化を行う

ことを特徴とする請求項１８記載のコンパイラ装置。

【請求項２１】　前記繰り返し回数に関する指定は、前記ループ処理が偶数回だけ繰り返される旨の保証であり、

前記最適化手段は、前記指示取得手段によって検出された指定の対象となったループ処理については、偶数回だけ繰り返されるものとして、ループアンローリ

ングによる最適化を行う

ことを特徴とする請求項１８記載のコンパイラ装置。

【請求項２２】　前記繰り返し回数に関する指定は、前記ループ処理が奇数回だけ繰り返される旨の保証であり、

前記最適化手段は、前記指示取得手段によって検出された指定の対象となったループ処理については、奇数回だけ繰り返されるものとして、ループアンローリングによる最適化を行う

ことを特徴とする請求項１８記載のコンパイラ装置。

【請求項２３】　高級言語で記述されたソースプログラムが記録されたコンピュータ読み取り可能な記録媒体であって、

前記ソースプログラムには、当該ソースプログラムを機械語プログラムに翻訳するコンパイラに対して、（１）特定のループ処理について、ループアンローリングによる最適化をする旨を指示する記述、（２）特定のループ処理について、ループアンローリングによる最適化をしない旨を指示する記述、及び、（３）特定のループ処理の繰り返し回数に関する保証を示す記述の少なくとも１つが含まれる

ことを特徴とする記録媒体。

【請求項２４】　ソースプログラムを機械語プログラムに翻訳するコンパイラ装置であって、

生成する機械語プログラムを最適化する旨の指示を取得する指示取得手段と、

取得された指示に従って機械語命令列を生成することにより、最適化を施す最適化手段とを備え、

前記最適化手段は、前記指示取得手段がｉｆ変換についての指示を取得した場合には、取得した指示に従ってｉｆ変換に関する最適化を施す

ことを特徴とするコンパイラ装置。

【請求項２５】　前記指示取得手段は、前記ソースプログラムを翻訳する旨の指示とともに、ｉｆ変換をしない旨の指示を取得し、

前記最適化手段は、前記指示取得手段がｉｆ変換をしない旨の指示を取得した場合に、前記ソースプログラム中の全てのｉｆ構造文について、ｉｆ変換が行わ

れることを抑制する

ことを特徴とする請求項２４記載のコンパイラ装置。

【請求項２６】　前記指示取得手段は、前記ソースプログラムの中において、特定のｉｆ構造文について、ｉｆ変換をする旨の指示を検出し、

前記最適化手段は、前記指示取得手段によって検出された指示の対象となったｉｆ構造文について、ｉｆ変換をする

ことを特徴とする請求項２４記載のコンパイラ装置。

【請求項２７】　前記指示取得手段は、前記ソースプログラムの中において、特定のｉｆ構造文について、ｉｆ変換をしない旨の指示を検出し、

前記最適化手段は、前記指示取得手段によって検出された指示の対象となったｉｆ構造文については、ｉｆ変換が行われることを抑制する

ことを特徴とする請求項２４記載のコンパイラ装置。

【請求項２８】　高級言語で記述されたソースプログラムが記録されたコンピュータ読み取り可能な記録媒体であって、

前記ソースプログラムには、当該ソースプログラムを機械語プログラムに翻訳するコンパイラに対して、（１）特定のｉｆ構造文について、ｉｆ変換をする旨を指示する記述、及び、（２）特定のｉｆ構造文について、ｉｆ変換をしない旨を指示する記述の少なくとも１つが含まれる

ことを特徴とする記録媒体。

【請求項２９】　ソースプログラムを機械語プログラムに翻訳するコンパイラ装置であって、

生成する機械語プログラムを最適化する旨の指示を取得する指示取得手段と、

取得された指示に従って機械語命令列を生成することにより、最適化を施す最適化手段とを備え、

前記最適化手段は、メモリ領域に配置する配列データのアライメントについての指示を取得した場合には、取得した指示に従ってデータをメモリ領域に配置することにより、前記最適化を施す

ことを特徴とするコンパイラ装置。

【請求項３０】　前記指示取得手段は、前記ソースプログラムを翻訳する旨

の指示とともに、特定の型の配列データのアライメントについての指定を取得し、

前記最適化手段は、前記ソースプログラムで宣言された前記特定の型の全ての配列データについて、その先頭アドレスが前記アライメントとなるようにメモリ領域に配置する

ことを特徴とする請求項２９記載のコンパイラ装置。

【請求項３１】　前記指示取得手段は、前記ソースプログラムを翻訳する旨の指示とともに、構造体のアライメントについての指定を取得し、

前記最適化手段は、前記ソースプログラムで宣言された前記構造体データについて、その先頭アドレスが前記アライメントとなるようにメモリ領域に配置する

ことを特徴とする請求項２９記載のコンパイラ装置。

【請求項３２】　前記指示取得手段は、前記ソースプログラムの中において、特定の変数名で示される引数のポインタ変数が指すデータのアライメントについての指定を検出し、

前記最適化手段は、前記指示取得手段によって検出された指定の対象となったデータについては、指定されたアライメントでメモリ領域に配置されているものとして、前記最適化を施す

ことを特徴とする請求項２９記載のコンパイラ装置。

【請求項３３】　前記指示取得手段は、前記ソースプログラムの中において、特定の変数名で示されるローカルポインタ変数が指すデータのアライメントについての指定を検出し、

前記最適化手段は、前記指示取得手段によって検出された指定の対象となったデータについては、指定されたアライメントでメモリ領域に配置されているものとして、前記最適化を施す

ことを特徴とする請求項２９記載のコンパイラ装置。

【請求項３４】　前記最適化手段は、メモリ領域に配置した前記データにアクセスするメモリアクセス命令については、２以上のデータを同時に転送するペア命令を生成する

ことを特徴とする請求項３０～３３のいずれか１項に記載のコンパイラ装置。

【請求項３５】　高級言語で記述されたソースプログラムが記録されたコンピュータ読み取り可能な記録媒体であって、

前記ソースプログラムには、当該ソースプログラムを機械語プログラムに翻訳するコンパイラに対して、（１）特定の変数名で示される引数のポインタ変数が指すデータのアライメントを保証する旨の記述、及び、（２）特定の変数名で示されるローカルポインタ変数が指すデータのアライメントを保証する旨の記述の少なくとも１つが含まれる

ことを特徴とする記録媒体。

【請求項３６】　ソースプログラムを機械語プログラムに翻訳するコンパイラ装置のためのプログラムであって、

請求項１～４、６～１２、１４～２２、２４～２７、２９～３４のいずれか１項に記載のコンパイラ装置が備える全ての手段としてコンピュータに機能させる

ことを特徴とするプログラム。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、Ｃ言語等の高級言語で記述されたソースプログラムを機械語プログラムに変換するコンパイラに関し、特に、コンパイラによる最適化についての指示に関する。

【０００２】

【従来の技術】

従来のコンパイラは、主に制御処理アプリケーション向けに作られている。制御処理アプリケーションでは、精密な実行性能及びコードサイズのチューニングはそれほど必要とされておらず、むしろ開発工数削減の観点から、ユーザは、「性能優先」、「コードサイズ優先」、あるいは、「性能とコードサイズのバランス」といった大雑把な指示（コンパイル時に指定するオプション等）のみ与えて、その最適化戦略のほとんどをコンパイラに任せている。

【０００３】

一方、クリティカルな実行性能及びコードサイズが要求されるメディア処理

アプリケーションの分野では、アセンブリ言語によるハンドコーディングを行うことにより、要求性能及びコードサイズを実現することを第一目標に開発が行われている。

【０００４】

【発明が解決しようとする課題】

しかしながら、近年、メディア処理アプリケーションの増大化、多様化により、開発工数が増大し、メディア処理分野においても高級言語によるアプリケーション開発が必要とされるようになってきている。そのために、高級言語によるメディア処理アプリケーション開発を実現する試みが行われている。その際、ユーザは、高級言語開発であっても、より精密なチューニングができることを期待しており、コンパイラが行う最適化戦略を詳細に制御することが必要となる。

【０００５】

したがって、従来のような大雑把な指示ではなく、コンパイラによる各種最適化の種類ごとにＯＮ／ＯＦＦやその程度を指定したり、プログラム中の変数やループ処理等の単位で最適化をＯＮ／ＯＦＦさせたりする等のきめ細かい制御が必要とされる。

【０００６】

そこで、本発明は、このような状況に鑑みてなされたものであり、コンパイラによる最適化をユーザが緻密に制御することが可能な柔軟性の高いコンパイラを提供することを目的とする。

【０００７】

【課題を解決するための手段】

上記目的を達成するために、本発明に係るコンパイラは、ソースプログラムを機械語プログラムに翻訳するコンパイラ装置であって、生成する機械語プログラムを最適化する旨の指示を取得する指示取得手段と、取得された指示に従って機械語命令列を生成することにより、最適化を施す最適化手段とを備え、前記最適化手段は、前記指示取得手段がグローバルメモリ領域に配置する配列データについての指示を取得した場合には、取得した指示に従ってグローバルメモリ領域に配置する配列データを決定することにより、前記最適化を施すことを特徴とする

【０００８】

また、本発明に係るコンパイラは、ソースプログラムを機械語プログラムに翻訳するコンパイラ装置であって、生成する機械語プログラムを最適化する旨の指示を取得する指示取得手段と、取得された指示に従って機械語命令列を生成することにより、最適化を施す最適化手段とを備え、前記最適化手段は、前記指示取得手段がソフトウェアパイプライニングによる最適化についての指示を取得した場合には、取得した指示に従ってソフトウェアパイプライニングに関する最適化を施すことを特徴とする。

【０００９】

また、本発明に係るコンパイラは、ソースプログラムを機械語プログラムに翻訳するコンパイラ装置であって、生成する機械語プログラムを最適化する旨の指示を取得する指示取得手段と、取得された指示に従って機械語命令列を生成することにより、最適化を施す最適化手段とを備え、前記最適化手段は、前記指示取得手段がループアンローリングによる最適化についての指示を取得した場合には、取得した指示に従ってループアンローリングに関する最適化を施すことを特徴とする。

【００１０】

また、本発明に係るコンパイラは、ソースプログラムを機械語プログラムに翻訳するコンパイラ装置であって、生成する機械語プログラムを最適化する旨の指示を取得する指示取得手段と、取得された指示に従って機械語命令列を生成することにより、最適化を施す最適化手段とを備え、前記最適化手段は、前記指示取得手段がｉｆ変換についての指示を取得した場合には、取得した指示に従ってｉｆ変換に関する最適化を施すことを特徴とする。

【００１１】

また、本発明に係るコンパイラは、ソースプログラムを機械語プログラムに翻訳するコンパイラ装置であって、生成する機械語プログラムを最適化する旨の指示を取得する指示取得手段と、取得された指示に従って機械語命令列を生成することにより、最適化を施す最適化手段とを備え、前記最適化手段は、メモリ領域

に配置する配列データのアライメントについての指示を取得した場合には、取得した指示に従ってデータをメモリ領域に配置することにより、前記最適化を施すことを特徴とする。

【００１２】

なお、本発明は、上記のようなコンパイラ装置として実現することができるだけでなく、このようなコンパイラ装置が備える手段をステップとするプログラムとして実現したり、コンパイラへの指示が含まれたソースプログラムとして実現することもできる。そして、そのようなプログラムは、ＣＤ－ＲＯＭ等の記録媒体やインターネット等の伝送媒体を介して広く流通させることができるのは言うまでもない。

【００１３】

【発明の実施の形態】

以下、本発明に係るコンパイラの実施の形態について図面を用いて詳細に説明する。

本実施の形態におけるコンパイラは、Ｃ言語等の高級言語で記述されたソースプログラムを特定のプロセッサが実行できる機械語プログラムに翻訳するクロスコンパイラであり、生成する機械語プログラムのコードサイズや実行時間に関する最適化の指示をきめ細かく指定することができるという特徴を有する。

【００１４】

まず、本実施の形態におけるコンパイラの対象となるプロセッサの一例について、図１～図３６を用いて説明する。

【００１５】

本実施の形態におけるコンパイラの対象となるプロセッサは、例えば、通常のマイコンに比べて実行可能な命令の並列性が高く、ＡＶメディア系信号処理技術分野をターゲットとして開発された汎用プロセッサである。

【００１６】

図１は、そのようなプロセッサの概略ブロック図の一例である。このプロセッサ１は、命令制御部１０、デコード部２０、レジスタファイル３０、演算部４０、Ｉ／Ｆ部５０、命令メモリ部６０、データメモリ部７０、拡張レジスタ部８０

及びＩ／Ｏインターフェース部９０から構成される。演算部４０は、ＳＩＭＤ型命令の演算を実行する算術論理・比較演算器４１～４３、乗算・積和演算器４４、バレルシフタ４５、除算器４６及び変換器４７からなる。乗算・積和演算器４４は、ビット精度を落とさないように、最長で６５ビットで累算する。また、乗算・積和演算器４４は、算術論理・比較演算器４１～４３と同様、ＳＩＭＤ型命令の実行が可能である。更に、このプロセッサ１は、算術論理・比較演算命令が最大３並列実行可能である。

【００１７】

図２は、算術論理・比較演算器４１～４３の概略図を示す。算術論理・比較演算器４１～４３それぞれは、ＡＬＵ部４１ａ、飽和処理部４１ｂ及びフラグ部４１ｃから構成される。ＡＬＵ部４１ａは、算術演算器、論理演算器、比較器、ＴＳＴ器からなる。対応する演算データのビット幅は、８ビット(演算器を４並列で使用)、１６ビット(演算器を２並列で使用)、３２ビットである(全演算器で３２ビットデータ処理)。更に算術演算結果に対しては、フラグ部４１ｃ等により、オーバーフローの検出とコンディションフラグの生成が行われる。各演算器、比較器、ＴＳＴ器の結果は、算術右シフト、飽和処理部４１ｂによる飽和、最大・最小値検出、絶対値生成処理が行われる。

【００１８】

図３は、バレルシフタ４５の構成を示すブロック図である。バレルシフタ４５は、セレクタ４５ａ、４５ｂ、上位バレルシフタ４５ｃ、下位バレルシフタ４５ｄ及び飽和処理部４５ｅから構成され、データの算術シフト(２の補数体系のシフト)または、論理シフト(符号なしシフト)を実行する。通常は、３２ビットもしくは、６４ビットのデータを入出力としている。レジスタ３０ａ、３０ｂに格納された被シフトデータに対して、別のレジスタまたは即値でシフト量が指定される。データは、左６３ビット～右６３ビットの算術または論理シフトが行われ、入力ビット長で出力される。

【００１９】

また、バレルシフタ４５は、ＳＩＭＤ型命令に対して、８、１６、３２、６４ビットのデータをシフトすることができる。例えば、８ビットデータのシフトを

４並列で処理することができる。

【００２０】

算術シフトは、２の補数体系のシフトであり、加算や減算時の小数点の位置合わせや、２のべき乗の乗算(２、２の２乗、２の（－１）乗、２の（－２）乗倍など)等のために行われる。

【００２１】

図４は、変換器４７の構成を示すブロック図である。変換器４７は、飽和ブロック(SAT)４７ａ、BSEQブロック４７ｂ、MSKGENブロック４７ｃ、VSUMBブロック４７、BCNTブロック４７ｅ及びILブロック４７ｆから構成される。

【００２２】

飽和ブロック(SAT)４７ａは、入力データに対する飽和処理を行う。３２ビットデータを飽和処理するブロックを２つ持つことにより、２並列のＳＩＭＤ型命令をサポートする。

【００２３】

BSEQブロック４７ｂは、MSBから連続する０か１をカウントする。

MSKGENブロック４７ｃは、指定されたビット区間を１、それ以外を０として出力する。

VSUMBブロック４７ｄは、入力データを指定されたビット幅に区切り、その総和を出力する。

BCNTブロック４７ｅは、入力データで１となっているビットの数をカウントする。

ILブロック４７ｆは、入力データを指定されたビット幅に区切り、各データブロックを入れ換えた値を出力する。

【００２４】

図５は、除算器４６の構成を示すブロック図である。除算器４６は、被除数を６４ビット、除数を３２ビットとし、商と剰余を３２ビットずつ出力する。商と剰余を求めるまでに３４サイクルを必要とする。符号付き、符号なし、両方のデータを扱うことが可能である。ただし、被除数と除数において符号の有無の設定は共通とする。その他、オーバーフローフラグ、０除算フラグを出力する機能を

有する。

【００２５】

図６は、乗算・積和演算器４４の構成を示すブロック図である。乗算・積和演算器４４は、２つの３２ビット乗算器（MUL）４４ａ、４４ｂ、３つの６４ビット加算器（Adder）４４ｃ～４４ｅ、セレクタ４４ｆ及び飽和処理部（Saturation）４４ｇから構成され、以下の乗算、積和演算を行う。

・３２×３２ビットのsignedの乗算、積和、積差演算

・３２×３２ビットのunsignedの乗算

・１６×１６ビットの２並列のsignedの乗算、積和、積差演算

・３２×１６ビットの２並列のsignedの乗算、積和、積差演算

【００２６】

これらの演算を整数、固定小数点フォーマット（ｈ１、ｈ２、ｗ１、ｗ２）のデータに対して行う。また、これらの演算に対し、丸め、飽和を行う。

【００２７】

図７は、命令制御部１０の構成を示すブロック図である。命令制御部１０は、命令キャッシュ１０ａ、アドレス管理部１０ｂ、命令バッファ１０ｃ～１０ｅ、ジャンプバッファ１０ｆ及びローテーション部（rotation）１０ｇから構成され、通常時及び分岐時の命令供給を行う。１２８ビットの命令バッファを３つ（命令バッファ１０ｃ～１０ｅ）持つことにより、最大並列実行数に対応している。分岐処理に関しては、分岐実行前に、ジャンプバッファ１０ｆ等を介して、後述するＴＡＲレジスタに予め分岐先アドレスを格納する(settar命令)。ＴＡＲレジスタに格納された分岐先アドレスを使用して、分岐を行う。

【００２８】

なお、本実施の形態におけるコンパイラの対象となるプロセッサ１は、例えば、ＶＬＩＷアーキテクチャを持つプロセッサである。ここで、ＶＬＩＷアーキテクチャとは、１つの命令語中に複数の命令(ロード、ストア、演算、分岐など)を格納し、それらを全て同時に実行するアーキテクチャである。プログラマは、並列実行可能な命令を１つの発行グループとして記述することによって、その発行グループを並列処理させることができる。本明細書では、発行グループの区切り

を";;"で示す。以下に表記例を示す。

（例１）

```
mov r1, 0x23;;
```

この命令記述は、命令movのみを実行することを意味する。

（例２）

```
mov r1, 0x38
add r0, r1, r2
sub r3, r1, r2;;
```

これらの命令記述は、命令mov、add、subを３並列で実行することを意味する。

【００２９】

命令制御部１０は、発行グループを識別し、デコード部２０に送る。デコード部２０では、発行グループの命令を解析し、必要な資源を制御する。

【００３０】

次に、このようなプロセッサ１が備えるレジスタについて説明する。

プロセッサ１のレジスタセットは、以下の表１に示される通りである。

【表１】

レジスタ名	ビット幅	本数	用途
R0～R31	32ビット	32本	汎用レジスタ。データメモリのポインタ、演算命令におけるデータの格納等に使用します。
TAR	32ビット	1本	分岐用レジスタ。分岐時の分岐アドレスの格納に使用します。
LR	32ビット	1本	リンク用レジスタ。
SVR	16ビット	2本	退避用レジスタ。コンディションフラグ(CFR)と各種モードを退避します。
M0～M1 (MH0:ML0～MH1:ML1)	64ビット	2本	演算用レジスタ。演算命令におけるデータの格納に使用します。

【００３１】

また、このようなプロセッサ１のフラグセット（後述する条件フラグレジスタ等で管理されるフラグ）は、以下の表２に示される通りである。

【表２】

C0～C7	1ビット	8本	条件フラグ。条件成立・不成立を表します。
VC0～VC3	1ビット	4本	メディア処理拡張命令用条件フラグ。条件成立・不成立を表します。
OVS	1ビット	1本	オーバーフローフラグ。演算時のオーバーフローを検出します。
CAS	1ビット	1本	キャリーフラグ。演算時のキャリーを検出します。
BPO	5ビット	1本	ビット位置指定。マスク処理命令時に処理対象となるビット位置を指定します。
ALN	2ビット	1本	バイトアライン指定。
FXP	1ビット	1本	固定小数点演算モード。
UDR	32ビット	1本	未定義レジスタ

【００３２】

図８は、汎用レジスタ（Ｒ０～Ｒ３１）３０ａの構造を示す図である。汎用レジスタ（Ｒ０～Ｒ３１）３０ａは、実行対象となっているタスクのコンテキストの一部を構成し、データまたはアドレスを格納する３２ビットのレジスタ群である。なお、汎用レジスタＲ３０およびＲ３１は、それぞれグローバルポインタ、スタックポインタとして、ハードウェアが使用する。

【００３３】

図９は、リンクレジスタ（ＬＲ）３０ｃの構造を示す図である。なお、このリンクレジスタ（ＬＲ）３０ｃと関連して、このプロセッサ１は、図示されていない退避レジスタ（ＳＶＲ）も備える。リンクレジスタ（ＬＲ）３０ｃは、関数コール時のリターンアドレスを格納する３２ビットのレジスタである。なお、退避レジスタ（ＳＶＲ）は、関数コール時の条件フラグレジスタのコンディションフラグ（CFR.CF）を退避する１６ビットのレジスタである。リンクレジスタ（ＬＲ）３０ｃは、後述する分岐レジスタ（ＴＡＲ）と同様に、ループ高速化にも使用される。下位１ビットは常に０が読み出されるが、書き込み時には０を書き込む必要がある。

【００３４】

例えば、call(brl,jmpl)命令を実行した場合には、このプロセッサ１は、リンクレジスタ（ＬＲ）３０ｃに戻りアドレスを退避し、退避レジスタ（ＳＶＲ）にコンディションフラグ(CFR.CF)を退避する。また、jmp命令を実行した場合には、リンクレジスタ（ＬＲ）３０ｃから戻りアドレス(分岐先アドレス)を取り出し

、プログラムカウンタ（ＰＣ）を復帰させる。さらに、ret(jmpr)命令を実行した場合には、リンクレジスタ（ＬＲ）３０ｃから分岐先アドレス(戻りアドレス)を取り出し、プログラムカウンタ（ＰＣ）に格納(復帰)する。さらに、退避レジスタ（ＳＶＲ）からコンディションフラグを取り出し、条件フラグレジスタ（ＣＦＲ）３２のコンディションンフラグ領域CFR.CFに格納(復帰)する。

【００３５】

図１０は、分岐レジスタ（ＴＡＲ）３０ｄの構造を示す図である。分岐レジスタ（ＴＡＲ）３０ｄは、分岐ターゲットアドレスを格納する３２ビットのレジスタである。主に、ループの高速化に用いられる。下位１ビットは常に０が読み出されるが、書き込み時には０を書き込む必要がある。

【００３６】

例えば、jmp,jloop命令を実行した場合には、プロセッサ１は、分岐レジスタ（ＴＡＲ）３０ｄから分岐先アドレスを取り出し、プログラムカウンタ（ＰＣ）に格納する。分岐レジスタ（ＴＡＲ）３０ｄに格納されたアドレスの命令が分岐用命令バッファに格納されている場合は、分岐ペナルティが０になる。分岐レジスタ（ＴＡＲ）３０ｄにループの先頭アドレスを格納しておくことでループを高速化することができる。

【００３７】

図１１は、プログラム状態レジスタ（ＰＳＲ）３１の構造を示す図である。プログラム状態レジスタ（ＰＳＲ）３１は、実行対象となっているタスクのコンテキストの一部を構成し、以下に示されるプロセッサ状態情報を格納する３２ビットのレジスタである。

【００３８】

ビットＳＷＥ：ＶＭＰ（Virtual Multi-Processor）のＬＰ（Logical Processor）切替えイネーブルを示す。「０」はＬＰ切替え不許可を示し、「１」はＬＰ切替え許可を示す。

【００３９】

ビットＦＸＰ：固定小数点モードを示す。「０」はモード０を示し、「１」はモード１を示す。

ビットＩＨ：割込み処理フラグであり、マスカブル割込み処理中であることを示す。「１」は割込み処理中であることを示し、「０」は割込み処理中でないことを示す。割込みが発生すると自動的にセットされる。rti命令で割込みから復帰したところが、他の割込み処理中かプログラム処理中であるのかを見分けるために使用される。

【００４０】

ビットＥＨ：エラーまたはＮＭＩを処理中であることを示すフラグである。「０」はエラー/ＮＭＩ割込み処理中でないことを示し、「１」はエラー/ＮＭＩ割込み処理中であることを示す。ＥＨ＝１のとき、非同期エラーまたはＮＭＩが発生した場合は、マスクされる。また、ＶＭＰイネーブル時はＶＭＰのプレート切り替えがマスクされる。

【００４１】

ビットＰＬ［１：０］：特権レベルを示す。「００」は特権レベル０、つまり、プロセッサアブストラクションレベルを示し、「０１」は特権レベル１（設定できない）を示し、「１０」は特権レベル２、つまり、システムプログラムレベルを示し、「１１」は特権レベル３、つまり、ユーザプログラムレベルを示す。

【００４２】

ビットＬＰＩＥ３：ＬＰ固有割込み３イネーブルを示す。「１」は割込み許可を示し、「０」は割込み不許可を示す。

ビットＬＰＩＥ２：ＬＰ固有割込み２イネーブルを示す。「１」は割込み許可を示し、「０」は割込み不許可を示す。

ビットＬＰＩＥ１：ＬＰ固有割込み１イネーブルを示す。「１」は割込み許可を示し、「０」は割込み不許可を示す。

ビットＬＰＩＥ０：ＬＰ固有割込み０イネーブルを示す。「１」は割込み許可を示し、「０」は割込み不許可を示す。

ビットＡＥＥ：ミスアライメント例外イネーブルを示す。「１」はミスアライメント例外許可を示し、「０」はミスアライメント例外不許可を示す。

ビットＩＥ：レベル割込みイネーブルを示す。「１」はレベル割込み許可を示し、「０」はレベル割込み不許可を示す。

ビットＩＭ［７：０］：割込みマスクを示す。レベル０～７まで定義され、個々のレベルでマスクすることができる。レベル０が最も高いレベルとなる。ＩＭによりマスクされていない割込み要求のうち最も高いレベルを持った割込み要求のみがプロセッサ１に受理される。割込み要求を受理すると受理したレベル以下のレベルはハードウェアで自動的にマスクされる。IM[0]はレベル0のマスクであり、IM[1]はレベル1のマスクであり、IM[2]はレベル2のマスクであり、IM[3]はレベル3のマスクであり、IM[4]はレベル4のマスクであり、IM[5]はレベル5のマスクであり、IM[6]はレベル6のマスクであり、IM[7]はレベル7のマスクである。

ｒｅｓｅｒｖｅｄ：予約ビットを示す。常に０が読み出される。書き込む時は０を書き込む必要がある。

【００４３】

図１２は、条件フラグレジスタ（ＣＦＲ）３２の構造を示す図である。条件フラグレジスタ（ＣＦＲ）３２は、実行対象となっているタスクのコンテキストの一部を構成する３２ビットのレジスタであり、コンディションフラグ(条件フラグ)、オペレーションフラグ(演算フラグ)、ベクタコンディションフラグ(ベクタ条件フラグ)、演算命令用ビット位置指定フィールド、SIMDデータアライン情報フィールドから構成される。

【００４４】

ビットＡＬＮ［１：０］：アラインモードを示す。valnvc命令のアラインモードを設定する。

ビットＢＰＯ［４：０］：ビットポジションを示す。ビット位置指定の必要な命令で使用する。

ビットＶＣ０～ＶＣ３：ベクタ条件フラグである。ＬＳＢ側のバイトあるいはハーフワードから順にＶＣ０に対応し、ＭＳＢ側がＶＣ３に対応する。

ビットＯＶＳ：オーバーフローフラグ(サマリー)である。飽和発生やオーバーフロー検出でセットされる。検出されなかった場合は、命令実行前の値を保持する。クリアはソフトで行う必要がある。

ビットＣＡＳ：キャリーフラグ(サマリー)である。addc命令でキャリーまたはsubc命令でボローが発生した場合セットされる。addc命令でキャリーもしくはsu

bc命令でボローが発生しなかった場合は、命令実行前の値を保持する。クリアはソフトで行う必要がある。

ビットＣ０～Ｃ７：コンディションフラグである。条件付き実行命令における条件（ＴＲＵＥ／ＦＡＬＳＥ）を示す。条件付き命令の条件とビットＣ０～Ｃ７との対応は、命令に含まれるプレディケート・ビットによって決定される。なお、フラグＣ７は常に値が１である。フラグＣ７へのＦＡＬＳＥ条件の反映(０書き込み)は無視される。

ｒｅｓｅｒｖｅｄ：予約ビットを示す。常に０が読み出される。書き込む時は０を書き込む必要がある。

【００４５】

図１３は、アキュムレータ（Ｍ０，Ｍ１）３０ｂの構造を示す図である。このアキュムレータ（Ｍ０，Ｍ１）３０ｂは、実行対象となっているタスクのコンテキストの一部を構成し、図１３（ａ）に示される３２ビットレジスタMH0-MH1（乗除算・積和用レジスタ(上位３２ビット)）と、図１３（ｂ）に示される３２ビットレジスタML0-ML1乗除算・積和用レジスタ(下位３２ビット)とからなる。

【００４６】

レジスタMH0-MHは、乗算命令では結果の上位３２ビットを格納するのに使用される。積和命令ではアキュムレータの上位３２ビットとして使用される。また、ビットストリームを取り扱う場合に汎用レジスタと組み合わせて使用することができる。レジスタML0-ML1は、乗算命令では結果の下位３２ビットを格納するのに使用される。積和命令ではアキュムレータの下位３２ビットとして使用される。

【００４７】

図１４は、プログラムカウンタ（ＰＣ）３３の構造を示す図である。このプログラムカウンタ（ＰＣ）３３は、実行対象となっているタスクのコンテキストの一部を構成し、実行中の命令のアドレスを保持する３２ビットのカウンタである。下位１ビットは常に０が格納される。

【００４８】

図１５は、ＰＣ退避用レジスタ（ＩＰＣ）３４の構造を示す図である。このＰ

Ｃ退避用レジスタ（ＩＰＣ）３４は、実行対象となっているタスクのコンテキストの一部を構成する３２ビットのレジスタであり、下位１ビットは常に０が読み出されるが、書き込み時には０を書き込む必要がある。

【００４９】

図１６は、ＰＳＲ退避用レジスタ（ＩＰＳＲ）３５の構造を示す図である。このＰＳＲ退避用レジスタ（ＩＰＳＲ）３５は、実行対象となっているタスクのコンテキストの一部を構成し、プログラム状態レジスタ（ＰＳＲ）３１を退避するための３２ビットのレジスタであり、プログラム状態レジスタ（ＰＳＲ）３１の予約ビットに対応する部分は常に０が読み出されるが、書き込み時には０を書き込む必要がある。

【００５０】

次に、本実施の形態におけるコンパイラの対象となるプロセッサ１のメモリ空間について説明する。例えば、プロセッサ１では、４ＧＢのリニアなメモリ空間を３２分割し、１２８ＭＢ単位の空間に命令ＳＲＡＭ（Static RAM）とデータＳＲＡＭが割り当てられる。この１２８ＭＢの空間を１ブロックとして、ＳＡＲ(ＳＲＡＭ Area Register)にアクセスしたいブロックを設定する。アクセスされたアドレスがＳＡＲで設定された空間である場合は、直接命令ＳＲＡＭ/データＳＲＡＭに対してアクセスを行うが、ＳＡＲで設定された空間でない場合は、バスコントローラ（ＢＣＵ）に対してアクセス要求を出する。ＢＣＵにはオン・チップ・メモリ（ＯＣＭ）、外部メモリ、外部デバイス、Ｉ／Ｏポート等が接続されており、それらのデバイスに対して読み書きを行うことができる。

【００５１】

図１７は、本実施の形態におけるコンパイラの対象となるプロセッサ１のパイプライン動作を示すタイミング図である。このプロセッサ１は、本図に示されるように、例えば、基本的に命令フェッチ、命令割り当て(ディスパッチ)、デコード、実行、書き込みの５段パイプラインで構成されている。

【００５２】

図１８は、このようなプロセッサ１による命令実行時の各パイプライン動作を示すタイミング図である。命令フェッチステージでは、プログラムカウンタ（Ｐ

Ｃ）３３で指定されるアドレスの命令メモリをアクセスし、命令を命令バッファ１０ｃ～１０ｅ等に転送する。命令割り当てステージでは、分岐系命令に対する分岐先アドレス情報の出力、入力レジスタ制御信号の出力、可変長命令の割り当てを行い、命令をインストラクションレジスタ（ＩＲ）に転送する。デコードステージでは、ＩＲをデコード部２０に入力し、演算器制御信号、メモリアクセス信号を出力する。実行ステージでは、演算を実行、演算結果をデータメモリか汎用レジスタ（Ｒ０～Ｒ３１）３０ａに出力する。書き込みステージでは、データ転送、演算結果を汎用レジスタに格納する。

【００５３】

本実施の形態におけるコンパイラの対象となるプロセッサ１は、例えば、ＶＬＩＷアーキテクチャにより上記の処理を最高３並列で行うことができる。したがって、図１８に示された動作については、本プロセッサ１は、図１９に示されるタイミングで並列に実行する。

【００５４】

次に、以上のように構成されたプロセッサ１の命令セットの例について説明する。

【００５５】

以下の表３～表５は、本実施の形態におけるコンパイラの対象となるプロセッサ１が実行する命令をカテゴリー別に分類した表である。

【表３】

カテゴリー	演算器	命令オペコード
メモリ転送命令(ロード)	M	ld, ldh, ldhu, ldb, ldbu, ldp, ldhp, ldbp, ldbh, ldbuh, ldbhp, ldbuhp
メモリ転送命令(ストア)	M	st, sth, stb, stp, sthp, stbp, stbh, stbhp
メモリ転送命令(その他)	M	dpref, ldstb
外部レジスタ転送命令	M	rd, rde, wt, wte
分岐命令	B	br, brl, call, jmp, jmpl, jmpr, ret, jmpf, jloop, setbb, setlr, settar
ソフトウェア割込み命令	B	rti, pi0, pi0l, pi1, pi1l, pi2, pi2l, pi3, pi3l, pi4, pi4l, pi5, pi5l, pi6, pi6l, pi7, pi7l, sc0, sc1, sc2, sc3, sc4, sc5, sc6, sc7
VMP/割込み制御命令	B	intd, inte, vmpsleep, vmpsus, vmpswd, vmpswe, vmpwait
算術演算命令	A	abs, absvh, absvw, add, addarvw, addc, addmsk, adds, addsr, addu, addvh, addvw, neg, negvh, negvw, rsub, sladd, s2add, sub, subc, submsk, subs, subvh, subvw, max, min
論理演算命令	A	and, andn, or, sethi, xor, not
比較命令	A	cmpCC, cmpCCa, cmpCCn, cmpCCo, tstn, tstna, tstnn, tstno, tstz, tstza, tstzn, tstzo
転送命令	A	mov, movcf, mvclcas, mvclovs, setlo, vcchk
NOP命令	A	nop
シフト命令1	S1	asl, aslvh, aslvw, asr, asrvh, asrvw, lsl, lsr, rol, ror
シフト命令2	S2	aslp, aslpvw, asrp, asrpvw, lslp, lsrp

【表４】

カテゴリー	演算器	命令オペコード
抽出命令	S2	ext, extb, extbu, exth, exthu, extr, extru, extu
マスク命令	C	msk, mskgen
飽和命令	C	sat12, sat9, satb, satbu, sath, satw
変換命令	C	valn, valn1, valn2, valn3, valnvc1, valnvc2, valnvc3, valnvc4, vhpkb, vhpkh, vhunpkb, vhunpkh, vintlhb, vintlhh, vintllb, vintllh, vlpkb, vlpkbu, vlpkh, vlpkhu, vlunpkb, vlunpkbu, vlunpkh, vlunpkhu, vstovb, vstovh, vunpk1, vunpk2, vxchngh, vexth
ビットカウント命令	C	bcnt1, bseq, bseq0, bseq1
その他	C	byterev, extw, mskbrvb, mskbrvh, rndvh, movp
乗算命令1	X1	fmulhh, fmulhhr, fmulhw, fmulhww, hmul, lmul
乗算命令2	X2	fmulww, mul, mulu
積和命令1	X1	fmachh, fmachhr, fmachw, fmachww, hmac, lmac
積和命令2	X2	fmacww, mac
積差命令1	X1	fmsuhh, fmsuhhr, fmsuhw, fmsuww, hmsu, lmsu
積差命令2	X2	fmsuww, msu
割算命令	DIV	div, divu
デバッガ命令	DBGM	dbgm0, dbgm1, dbgm2, dbgm3

【表５】

カテゴリー	演算器	命令オペコード
SIMD 算術演算命令	A	vabshvh, vaddb, vaddh, vaddhvc, vaddhvh, vaddrhvc, vaddsb, vaddsh, vaddsrb, vaddsrh, vasubb, vcchk, vhaddh, vhaddhvh, vhsubh, vhsubhvh, vladdh, vladdhvh, vlsubh, vlsubhvh, vnegb, vnegh, vneghvh, vsaddb, vsaddh, vsgnh, vsrsubb, vsrsubh, vssubb, vssubh, vsubb, vsubh, vsubhvh, vsubsh, vsumh, vsumh2, vsumrh2, vxaddh, vxaddhvh, vxsubh, vxsubhvh, vmaxb, vmaxh, vminb, vminh, vmovt, vsel
SIMD 比較命令	A	vcmpeqb, vcmpeqh, vcmpgeb, vcmpgeh, vcmpgtb, vcmpgth, vcmpleb, vcmpleh, vcmpltb, vcmplth, vcmpneb, vcmpneh, vscmpeqb, vscmpeqh, vscmpgeb, vscmpgeh, vscmpgtb, vscmpgth, vscmpleb, vscmpleh, vscmpltb, vscmplth, vscmpneb, vscmpneh
SIMD シフト命令1	S1	vaslb, vaslh, vaslvh, vasrb, vasrh, vasrvh, vislb, vislh, visrb, visrh, vrolb, vrolh, vrorb, vrorh
SIMD シフト命令2	S2	vasl, vaslvw, vasr, vasrvw, visl, visr
SIMD 飽和命令	C	vsath, vsath12, vsath8, vsath8u, vsath9
SIMD その他の命令	C	vabssumb, vrndvh
SIMD 乗算命令	X2	vfmulh, vfmulhr, vfmulw, vhfmulh, vhfmulhr, vhfmulw, vhmul, vlfmulh, vlfmulhr, vlfmulw, vlmul, vmul, vpfmulhww, vxfmulh, vxfmulhr, vxfmulw, vxmul
SIMD 積和命令	X2	vfmach, vfmachr, vfmacw, vhfmach, vhfmachr, vhfmacw, vhmac, vlfmach, vlfmachr, vlfmacw, vlmac, vmac, vpfmachww, vxfmach, vxfmachr, vxfmacw, vxmac
SIMD 積差命令	X2	vfmsuh, vfmsuw, vhfmsuh, vhfmsuw, vhmsu, vlfmsuh, vlfmsuw, vlmsu, vmsu, vxfmsuh, vxfmsuw, vxmsu

【００５６】

なお、表中の「演算器」は、その命令が使用する演算器を示す。演算器の略号の意味は次の通りである。つまり、「Ａ」はＡＬＵ命令、「Ｂ」は分岐命令、「Ｃ」は変換命令、「ＤＩＶ」は除算命令、「ＤＢＧＭ」はデバッグ命令、「Ｍ」はメモリアクセス命令、「Ｓ１」、「Ｓ２」はシフト命令、「Ｘ１」、「Ｘ２」は乗算命令を意味する。

【００５７】

図２０は、このようなプロセッサ１が実行する命令のフォーマットの例を示す図である。そのフォーマットには、図２０（ａ）に示される１６ビット命令フォ

ーマットと、図２０（ｂ）に示される３２ビット命令フォーマットとがある。

【００５８】

なお、図中における略号の意味は次の通りである。つまり、「Ｅ」はエンドビット（並列実行の境界）、「Ｆ」はフォーマットビット（００、０１、１０：１６ビット命令フォーマット、１１：３２ビット命令フォーマット）、「Ｐ」はプレディケート（実行条件：８個の条件フラグＣ０～Ｃ７のいずれかを指定）、「ＯＰ」はオペコードフィールド、「Ｒ」はレジスタフィールド、「Ｉ」は即値フィールド、「Ｄ」ディスプースメントフィールドを意味する。なお、「Ｅ」フィールドはＶＬＩＷに特有のもので、Ｅ＝０の命令は次の命令と並列に実行される。つまり、「Ｅ」フィールドによって並列度が可変のＶＬＩＷを実現している。

【００５９】

図２１～図３６は、プロセッサ１が実行する命令の概略的な機能を説明する図である。つまり、図２１は、カテゴリー「ALUadd（加算）系」に属する命令を説明する図であり、図２２は、カテゴリー「ALUsub（減算）系」に属する命令を説明する図であり、図２３は、カテゴリー「ALUlogic（論理演算）系ほか」に属する命令を説明する図であり、図２４は、カテゴリー「CMP（比較演算）系」に属する命令を説明する図であり、図２５は、カテゴリー「mul（乗算）系」に属する命令を説明する図であり、図２６は、カテゴリー「mac（積和演算）系」に属する命令を説明する図であり、図２７は、カテゴリー「msu（積差演算）系」に属する命令を説明する図であり、図２８は、カテゴリー「MEMld（メモリ読み出し）系」に属する命令を説明する図であり、図２９は、カテゴリー「MEMstore（メモリ書き出し）系」に属する命令を説明する図であり、図３０は、カテゴリー「BRA（分岐）系」に属する命令を説明する図であり、図３１は、カテゴリー「BSasl（算術バレルシフト）系ほか」に属する命令を説明する図であり、図３２は、カテゴリー「BSlsr（論理バレルシフト）系ほか」に属する命令を説明する図であり、図３３は、カテゴリー「CNVvaln（算術変換）系」に属する命令を説明する図であり、図３４は、カテゴリー「CNV（一般変換）系」に属する命令を説明する図であり、図３５は、カテゴリー「SATvlpk（飽和処理）系」に属する命令を説明する図であり、図３６は、カテゴリー「ETC（その他）系」に属する命

令を説明する図である。

【００６０】

これらの図において、項目「ＳＩＭＤ」は、その命令の型（ＳＩＳＤ（SINGLE）かＳＩＭＤかの区別）を示し、項目「サイズ」は、演算の対象となる個々のオペランドのサイズを示し、項目「命令」は、その命令のオペコードを示し、項目「オペランド」は、その命令のオペランドを示し、項目「ＣＦＲ」は、条件フラグレジスタの変化を示し、項目「ＰＳＲ」は、プロセッサ状態レジスタの変化を示し、項目「代表的な動作」は、動作の概要を示し、項目「演算器」は、使用される演算器を示し、項目「３１１６」は、命令のサイズを示す。

【００６１】

以下に、後述する具体例で使用される主な命令についてのプロセッサ１の動作を説明する。

ld Rb,(Ra,D10)

レジスタRaにディスプレースメント値(D10)を加算したアドレスからワードデータをレジスタRbにロードする。

ldh Rb,(Ra+)I9

レジスタRaが示すアドレスからハーフワードデータを符号拡張してロードする。さらに、レジスタRaに即値(I9)を加算し、レジスタRaに格納する。

ldp Rb:Rb+1,(Ra+)

レジスタRaが示すアドレスから レジスタRbとRb+1に2 つのワードデータを符号拡張してロードする。さらに、レジスタRaに 8 を加算し、レジスタRaに格納する。

【００６２】

ldhp Rb:Rb+1, (Ra+)

レジスタRaが示すアドレスから 2 つのハーフワードデータを符号拡張してロードする。さらに、レジスタRaに 4 を加算し、レジスタRaに格納する。

setlo Ra,I16

レジスタRaに即値(I16)を符号拡張して格納する。

sethi Ra,I16

レジスタRaの上位 16 bitに即値(I16)を格納する。レジスタRaの下位 16 bitには影響しない。

【００６３】

ld Rb,(Ra)

レジスタRaが示すアドレスからワードデータをレジスタRbにロードする。

add Rc,Ra,Rb

レジスタRaとRbを加算し、レジスタRbに格納する。

addu Rb,GP,I13

レジスタGPに即値(I13)を加算し、レジスタRbに格納する。

st (GP,D13),Rb

レジスタGPにディスプレースメント値(D13)を加算したアドレスに、レジスタRbに格納されたハーフワードデータをストアする。

【００６４】

sth (Ra+)I9,Rb

レジスタRaが示すアドレスに、レジスタRbに格納されたハーフワードデータをストアする。さらに、レジスタRaに即値(I9)を加算し、レジスタRaに格納する。

stp (Ra+),Rb:Rb+1

レジスタRaが示すアドレスに、レジスタRbとRb+1に格納された 2 つのワードデータをストアする。さらに、レジスタRaに 8 を加算し、レジスタRaに格納する。

ret

サブルーチンコールからのリターンに使用する。LRに格納されているアドレスに分岐する。SVR.CFをCFR.CFに転送する。

【００６５】

mov Ra,I16

レジスタRaに値(I16)を符号拡張して格納する。

settar C6,D9

以下の処理を行う。 (1) PCとディスプレースメント値(D9)を加算したアドレスを 分岐レジスタTARに格納する。 (2) そのアドレスの命令をフェッチして分

岐用命令バッファに格納する。（3）C6を 1 にセットする。

settar C6,Cm,D9

以下の処理を行う。（1）PCとディスプレースメント値(D9)を加算したアドレスを 分岐レジスタTARに格納する。（2）そのアドレスの命令をフェッチして分岐用命令バッファに格納する。（3）C6を 1 に、Cmを 0 にセットする。

【００６６】

settar C6,C2:C4,D9

以下の処理を行う。（1）PCとディスプレースメント値(D9)を加算したアドレスを 分岐レジスタTARに格納する。（2）そのアドレスの命令をフェッチして分岐用命令バッファに格納する。（3）C4とC6を 1 に、C2とC3を 0 にセットする。

jloop C6,TAR,Ra2,-1

ループで使用する。以下の処理を行う。（1）レジスタRa2に -1 を加算し、レジスタRa2に格納する。レジスタRa2が 0 より小さくなるとC6に 0 をセットする。（2）分岐レジスタTARが示すアドレスにジャンプする。

jloop C6,Cm,TAR,Ra2,-1

ループで使用する。以下の処理を行う。（1）Cm に 1 をセットする。（2）レジスタRa2に -1 を加算し、レジスタRa2に格納する。レジスタRa2が 0 より小さくなるとC6に 0 をセットする。（3）分岐レジスタTARが示すアドレスにジャンプする。

【００６７】

jloop C6,C2:C4,TAR,Ra2,-1

ループで使用する。以下の処理を行う。（1）C3をC2に転送し、C4をC3とC6に転送する。（2）レジスタRa2に -1 を加算し、レジスタRa2に格納する。レジスタRa2が 0 より小さくなるとC4に 0 をセットする。（3）分岐レジスタTARが示すアドレスにジャンプする。

mul Mm,Rb,Ra,I8

レジスタRaと即値(I8)を符号付乗算し、結果をレジスタMmとレジスタRbに格納する。

mac Mm,Rc,Ra,Rb,Mn

レジスタRaとRbを整数乗算し、レジスタMnと加算する。結果をレジスタMmとレジスタRcに格納する。

【００６８】

lmac Mm,Rc,Ra,Rb,Mn

レジスタRbをハーフワードベクタ形式で扱う。レジスタRaとRbの下位 16 bit を整数乗算し、レジスタMnと加算する。結果をレジスタMmとレジスタRcに格納する。

jloop C6,C2:C4,TAR,Ra2,-1

ループで使用する。以下の処理を行う。(1) C3をC2に転送し、C4をC3とC6に転送する。(2) レジスタRa2に -1 を加算し、レジスタRa2に格納する。レジスタRa2が 0 より小さくなるとC4に 0 をセットする。(3) 分岐レジスタTARが示すアドレスにジャンプする。

asr Rc,Ra,Rb

レジスタRaをRbが示すビット数だけ算術右シフトする。レジスタRbは±31 以内に飽和され、負の場合は算術左シフトになる。

br D9

現在のPCに、ディスプレースメント値(D9)を加算し、そのアドレスに分岐する。

【００６９】

jmpf TAR

分岐レジスタTARに格納されているアドレスに分岐する。

cmpCC Cm,Ra,I5

CC には次のCC比較条件を記述可能である。

eq/ne/gt/ge/gtu/geu/le/lt/leu/ltu

CCがeq/ne/gt/ge/le/ltの場合、I5は符号付きの値で、符号拡張して比較する。CCがgtu/geu/leu/ltuの場合、I5は符号なしの値である。

【００７０】

[コンパイラ]

次に、以上のプロセッサ１をターゲットとする本実施の形態におけるコンパイラについて説明する。

【００７１】

図３７は、本実施の形態におけるコンパイラ１００の構成を示す機能ブロック図である。このコンパイラ１００は、Ｃ言語等の高級言語で記述指定されたソースプログラム１０１を、上述のプロセッサ１をターゲットプロセッサとする機械語プログラム１０２に変換するクロスコンパイラであり、パーソナルコンピュータ等のコンピュータ上で実行されるプログラムによって実現され、大きく分けて、解析部１１０と、最適化部１２０と、出力部１３０とから構成される。

【００７２】

解析部１１０は、コンパイルの対象となるソースプログラム１０１及びこのコンパイラ１００に対するユーザからの指示等を字句解析することによって、コンパイラ１００に対する指示（オプション及びプラグマ）については最適化部１２０や出力部１３０に伝達し、コンパイルの対象となるプログラムについては内部形式データに変換したりする。

【００７３】

なお、「オプション」とは、コンパイラ１００を起動する際に、コンパイルの対象となるソースプログラム１０１の指定とともに、ユーザが任意に指定することができるコンパイラ１００への指示であり、生成する機械語プログラム１０２のコードサイズ及び実行時間を最適化するための指示等が含まれる。例えば、ユーザは、ソースプログラム１０１「sample.c」をコンパイルするときに、コマンド「ammmp-cc」を用いて、コンピュータ上で、

```
c:¥>ammmp-cc -o -max-gp-datasize=40 sample.c
```

と入力することができる。このコマンドにおける付加的な指示「-o」及び「-max-gp-datasize=40」がオプションである。このようなオプションによる指示は、ソースプログラム１０１全体に対する指示として扱われる。

【００７４】

また、「プラグマ（又は、プラグマ指令）」とは、ソースプログラム１０１中にユーザが任意に指定（配置）することができるコンパイラ１００への指示であ

り、オプションと同様に、生成する機械語プログラム１０２のコードサイズ及び実行時間を最適化するための指示等が含まれる。本実施の形態におけるコンパイラ１００では、「#pragma」で始まる文字列である。例えば、ユーザは、ソースプログラム１０１中に、

【００７５】

#pragma_no_gp_access　変数名

というステートメントを記述しておくことができる。このステートメントがプラグマ（プラグマ指令）である。このようなプラグマは、オプションと異なり、当該プラグマの直後に配置された変数やループ処理等だけに対する個別的な指示として扱われる。

【００７６】

最適化部１２０は、解析部１１０から出力されたソースプログラム１０１（内部形式データ）に対して、解析部１１０からの指示等に従って、（１）実行速度の向上を優先した最適化、（２）コードサイズの削減を優先した最適化、（３）実行速度とコードサイズの両方の最適化、の中から選択された最適化を実現するための全体的な最適化処理を行うことに加えて、ユーザによるオプション及びプラグマによって指定された個別的な最適化処理を行う処理部（グローバル領域割り付け部１２１、ソフトウェアパイプライニング部１２２、ループアンローリング部１２３、ｉｆ変換部１２４及びペア命令生成部１２５）を有する。

【００７７】

グローバル領域割り付け部１２１は、グローバル領域（共通のデータ領域として関数を超えて参照可能なメモリ領域）に配置する変数（配列）の最大データサイズの指定、グローバル領域に配置させる変数の指定、及び、グローバル領域に配置させない変数の指定に関するオプション及びプラグマに従った最適化処理を行う。

【００７８】

ソフトウェアパイプライニング部１２２は、ソフトウェアパイプライニングを行わない旨の指示、プロログ部・エピログ部が除去できる範囲でソフトウェアパイプライニングを行う旨の指示、及び、プロログ部・エピログ部を除去せずに可

能な範囲でソフトウェアパイプライニングを行う旨の指示に関するオプション及びプラグマに従った最適化処理を行う。

【００７９】

ループアンローリング部１２３は、ループアンローリングを行う旨の指示、ループアンローリングを行わない旨の指示、ループが繰り返される最低回数の保証、ループが偶数回繰り返される旨の保証、及び、ループが奇数回繰り返される旨の保証に関するオプション及びプラグマに従った最適化処理を行う。

【００８０】

ｉｆ変換部１２４は、ｉｆ変換を行う旨の指示、及び、ｉｆ変換を行わない旨の指示に関するオプション及びプラグマに従った最適化処理を行う。

【００８１】

ペア命令生成部１２５は、配列と構造体の先頭アドレスのアラインの指定、及び、関数引数のポインタ変数やローカルポインタ変数の指すデータのアライメントの保証に関するプラグマに従った最適化処理を行う。

【００８２】

出力部１３０は、最適化部１２０による最適化処理が施されたソースプログラム１０１に対して、内部形式データを対応する機械語命令に置き換えたり、ラベルやモジュール等のアドレスを解決したりすることで、機械語プログラム１０２を生成し、ファイル等として出力する。

【００８３】

次に、以上のように構成された本実施の形態におけるコンパイラ１００の特徴的な動作について具体例を示しながら説明する。

［グローバル領域割り付け部１２１］

まず、グローバル領域割り付け部１２１の動作とその意義について説明する。グローバル領域割り付け部１２１は、大きく分けて、（１）グローバル領域配置の最大データサイズの指定に関する最適化と、（２）グローバル領域配置の指定に関する最適化とを行う。

【００８４】

まず、（１）グローバル領域配置の最大データサイズの指定に関する最適化に

ついて説明する。

上記プロセッサ１には、グローバルポインタレジスタ（gp；汎用レジスタＲ３０）が用意されており、グローバル領域（以下ｇｐ領域とする）の先頭のアドレスを保持している。ｇｐ領域先頭からのディスプレースメントが最大１４ビットの範囲については、１命令でアクセスすることが可能である。

【００８５】

このｇｐ領域には、外部変数・静的変数等の配列を配置することが可能である。ただし、１命令でアクセスできる範囲を超えた場合、逆に性能が低下するため注意が必要である。

【００８６】

図３８（ａ）は、グローバル領域におけるデータ等の配置例を示す図である。ここでは、配列Aのデータサイズは、最大データサイズを超えない値であり、配列Cのデータサイズは、最大データサイズを超える値である。

【００８７】

ｇｐ領域に実体が収まっている配列Aへのアクセスは、以下の例のように、１命令で可能である。

例：ld　　r1,(gp,_A - .MN.gptop);;

なお、この例において、「.MN.gptop」は、グローバルポインタレジスタと同じアドレスを指すセクション名（ラベル）である。

【００８８】

一方、ｇｐ領域配置の最大データサイズを超える配列Cの場合は、ｇｐ領域以外に実体が配置され、配列Cのアドレスのみがｇｐ領域に配置される（なお、後述の、#pragma _no_gp_access指令を使用した場合は、ｇｐ領域に実体もアドレスも格納されない）。

【００８９】

この場合、配列Cへのアクセスは、以下の例のように、複数命令必要になる。

例：gpアドレス間接アクセスの場合

```
ld    r1,(gp,_C$ - .MN.gptop);;
ld    r1,(r1,8);;
```

例： 絶対アドレスアクセスの場合

```
setlo r0,LO(_C+8);;
sethi r0,HI(_C+8);;
ld    r0,(r0);;
```

【００９０】

なお、図３８（ｂ）に示されるグローバル領域以外の領域における配置例のように、ｇｐ領域の１命令でアクセスできる範囲外に実体が配置された配列Zの場合でも、以下のようなコードが生成される。

```
ld    r0,(gp,_Z - .MN.gptop);;
```

このコードは、１命令でアクセスできる範囲を超えているため、リンカにより複数命令に展開される。よって、１命令アクセスにはならない。

【００９１】

なお、ｇｐ領域の１命令アクセス範囲は、最大１４ビット範囲であるが、オブジェクトの型サイズにより、その範囲は異なる。つまり、８バイト型であれば１４ビット範囲であり、４バイト型であれば１３ビット範囲であり、２バイト型であれば１２ビット範囲であり、１バイト型であれば１１ビット範囲である。

【００９２】

コンパイラ１００は、最大データサイズ（デフォルト３２バイト）以下の配列・構造体の実体をｇｐ領域に配置する。一方、グローバル領域配置の最大データサイズを超えるオブジェクトに関しては、ｇｐ領域以外に実体を配置し、ｇｐ領域にはオブジェクトの先頭アドレスのみを配置する。

【００９３】

ここで、ｇｐ領域に余裕があれば、データサイズ３２バイト以上のオブジェクトも配置させた方が、より良いコードを生成することが可能である。

【００９４】

そこで、以下のオプションを用いることで、ユーザは、この最大データサイズを任意の値に指定することが可能となっている。

・コンパイルオプション

```
-mmax-gp-datasize=NUM
```

ここで、ＮＵＭは、グローバル領域に配置できる、一つの配列および構造体の最大データサイズの指定バイト（デフォルト３２バイト）である。

【００９５】

図３９は、グローバル領域割り付け部１２１の動作を示すフローチャートである。解析部１１０によって上記オプションが検出された場合には（ステップＳ１００、Ｓ１０１）、グローバル領域割り付け部１２１は、ソースプログラム１０１で宣言されている指定されたサイズ（ＮＵＭバイト）以下の全ての変数（配列）については、グローバル領域に配置し、ＮＵＭバイトを超える変数については、その先頭アドレスだけをグローバル領域に配置し、その実体をグローバル領域以外のメモリ領域に配置する（ステップＳ１０２）。このオプションによって、速度向上とサイズ削減という最適化が可能となる。

【００９６】

なお、この-mmax-gp-datasizeオプションでは、変数個々の配置指定はできない。個々の変数に対してｇｐ領域に配置する/しないを指定するには、後述する#pragma _gp_access指令を使用すればよい。また、外部変数・静的変数は、可能な限り１命令アクセス可能なｇｐ領域に配置することが好ましい。さらに、extern 宣言を使用して、他のファイルで定義される外部変数にアクセスする場合は、その外部変数のサイズを省略せずに、明記することが好ましい。例えば、外部変数の定義が、

```
int a[8];
```

とされている場合、使用するファイルでは、

```
extern int a[8];
```

と宣言することが好ましい。

【００９７】

なお、extern 宣言外部変数に #pragma _gp_access指令を使用する場合、定義の指定（配置する領域）と使用側の指定（アクセスする方法）を必ず合わせる必要がある。

【００９８】

次に、このようなオプション用いることによる最適化の具体例を示す。

図４０は、最大データサイズを変更した場合の最適化の具体例を示す図である。つまり、デフォルトの状態でコンパイルし、その結果、まだｇｐ領域に空きがあるので、以下のコマンド例のように、最大データサイズを変更してコンパイルした場合における両ケースで得られる生成コードの例が示されている。

```
c:¥>ammmp-cc -O -mmax-gp-datasize=40 sample.c
```

【００９９】

ここでは、配列ｃのオブジェクトサイズが、４０バイトであるとする。本図の左欄は、デフォルトの状態でコンパイルされた場合に生成されるコードの例であり、本図の右欄は、最大データサイズを４０に変更してコンパイルされた場合に生成されるコードの例である。なお、本図の最上段、中上段、中下段、最下段は、それぞれ、各欄のタイトル、サンプルプログラム（ソースプログラム１０１）、そこから生成されるコード（機械語プログラム１０２）、その生成コードのサイクル数及びコードサイズを示している（以下、最適化の具体例を示す他の図についても同様）。

【０１００】

本図の左欄の生成コードから分かるように、配列ｃの実体がｇｐ領域以外に配置されているため、複数命令での絶対アドレスアクセスになっている。一方、本図の右欄の生成コードから分かるように、配列ｃの実体がｇｐ領域に置かれるように最大データサイズ（40）が指定されたので、配列ｃの実体がｇｐ領域に配置され、１命令アクセスでのｇｐ相対アクセスになり、実行速度が向上される。つまり、デフォルトでは、１０サイクルで実行される８バイトのコードが生成されるのに対し、最大データサイズの変更によって、７サイクルで実行される５バイトのコードが生成される。

【０１０１】

もう一つの具体例として、ファイル外定義の外部変数の場合の具体例を図４１に示す。ここでは、ｇｐ領域配置の最大データサイズは４０であるとする。本図の左欄は、ファイル外定義の外部変数に関してサイズ指定がない場合に生成されるコードの例を示し、本図の右欄は、サイズ指定がある場合に生成されるコードの例を示している。

【０１０２】

本図の左欄に示されるように、外部定義配列ａと外部定義配列ｃのサイズが共に不明なので、ｇｐ領域に配置されているのか否かコンパイラ１００では判断できず、複数命令での絶対アドレスアクセスのコードが生成されている。

【０１０３】

一方、本図の右欄に示されるように、配列ａの定義サイズが４０バイト以下なので、ｇｐ領域に実体を配置し、グローバルポインタレジスタ（gp）を用いて１命令でのｇｐ相対アクセスのコードが生成されている。また、外部定義配列ｃのサイズも明示的に指定されており、ｇｐ領域配置の最大データサイズ以下であるため、配列ｃの実体がｇｐ領域に配置されているものとし、ｇｐ相対アクセスのコードが生成されている。このように、ファイル外定義の外部変数のサイズを指定しない場合には、１０サイクルで実行される１２バイトのコードが生成されるのに対し、最大データサイズの変更と外部変数のサイズ指定とを行った場合には、７サイクルで実行される５バイトのコードが生成される。

【０１０４】

次に、グローバル領域割り付け部１２１による、（２）グローバル領域配置の指定に関する最適化について説明する。

前述の、グローバル領域配置の最大データサイズ指定（ -mmax-gp-datasizeオプション）では、最大データサイズでのみ、ｇｐ領域の配置を指定するため、期待しない変数までｇｐ領域に配置されることがある。

【０１０５】

そこで、変数ごとにｇｐ領域の配置を指定する#pragma指令が用意されている。

・#pragma指令

#pragma _no_gp_access 変数名 [,変数名,...]

#pragma _gp_access 変数名 [,変数名,...]

ここで、[]内は省略可能を意味する。複数指定する場合、","（カンマ）で変数名を区切ればよい。なお、オプションとプラグマ指令とが重複又は矛盾した場合は、プラグマ指令が優先する。

【０１０６】

このようなプラグマ指令に対して、コンパイラ１００は次のように動作する。つまり、図３９において、解析部１１０によってプラグマ指令「#pragma _no_gp_access 変数名［,変数名,...］」が検出された場合には（Ｓ１００、Ｓ１０１）、グローバル領域割り付け部１２１は、ここで指定された変数については、オプション指定にかかわらず、グローバル領域に配置させないコードを生成し（ステップＳ１０３）、一方、解析部１１０によってプラグマ指令「#pragma _gp_access 変数名［,変数名,...］」が検出された場合には（Ｓ１００、Ｓ１０１）、グローバル領域割り付け部１２１は、ここで指定された変数については、オプション指定にかかわらず、グローバル領域に配置させるコードを生成する（ステップＳ１０４）。これらの#pragma指令によって、速度向上とサイズ削減という最適化が可能となる。

【０１０７】

なお、#pragma _no_gp_access指令が指定された場合は、グローバル領域割り付け部１２１は、その変数については、ｇｐ領域に実体もアドレスも配置しない。また、最大データサイズ指定よりも、#pragma _gp_access指令の方を優先する。もし、同一の変数に関して異なる指定が現れた場合には、コンパイラ１００の動作は不定となる。外部変数・静的変数については、可能な限り１命令アクセス可能なｇｐ領域に配置することが好ましい。

【０１０８】

次に、このようなプラグマ指令用いることによる最適化の具体例を示す。#pragma _gp_access指令を使用すると、ｇｐ領域配置の最大データサイズ以上の外部変数・静的変数をｇｐ領域に配置させることができので、その好適例を示す。

【０１０９】

図４２は、#pragma _no_gp_access指令を用いた場合に生成されるコードの例（左欄）と、#pragma _gp_access指令を用いた場合に生成されるコードの例（右欄）とを示す図である。

【０１１０】

本図の左欄に示されるように、配列ｃのサイズは４０バイトなのでデフォルト

の場合、先頭アドレスのみｇｐ領域に配置され、実体はｇｐ領域に配置されない。また、外部定義されている配列ａのサイズが３２バイトなのでデフォルトの場合は、ｇｐ領域に実体が配置されているとコンパイラ１００は判断する。

【０１１１】

しかし、#pragma _no_gp_access指令により、配列ｃについては、ｇｐ領域には先頭アドレスも実体も配置されずに、ｇｐ領域以外に実体が配置され、絶対アドレスアクセスのコードが生成される。外部定義の配列ａについても、ｇｐ領域以外に実体が配置されているとして、絶対アドレスアクセスのコードが生成される。

【０１１２】

一方、本図の右欄に示されるように、配列ｃはサイズが４０バイトなので、デフォルトの場合ではｇｐ領域に配置されないが、#pragma _gp_access指令により配列ｃの実体がｇｐ領域に配置される。ファイル外定義の配列ａは、サイズが不明であるが#pragma指令により、ｇｐ領域に配置されているものとし、ｇｐ相対アクセスコードが生成される。

【０１１３】

このように、#pragma _no_gp_access指令を用いた場合には、１０サイクルで実行される１２バイトのコードが生成されるのに対し、#pragma _gp_access指令を用いた場合には、７サイクルで実行される５バイトのコードが生成される。

【０１１４】

なお、extern 宣言外部変数に #pragma _gp_access指令を使用する場合、定義の指定（配置する領域）と使用側の指定（アクセスする方法）を必ず合わせておくことが好ましい。

【０１１５】

［ソフトウェアパイプライニング部１２２］

次に、ソフトウェアパイプライニング部１２２の動作とその意義について説明する。

ソフトウェアパイプライニング最適化は、ループ高速化手法の１つである。この最適化が行われると、ループ構造がプロログ部、カーネル部、 エピログ部に

変換される。なお、ソフトウェアパイプライニング最適化は、それによって実行速度が向上されると判断された場合に行われる。カーネル部は、各イタレーション（繰り返し）をその前後のイタレーションとオーバーラップさせる。これにより、１イタレーション毎の平均処理時間が削減される。

【０１１６】

図４３（ａ）は、ループ処理におけるプロログ部・カーネル部・エピログ部の概念図である。ここでは、イタレーション間で依存関係のない命令X、Y、Z が５回繰り返される場合の命令コード、実行イメージ、生成コードイメージの例が示されている。なお、ループ処理とは、for文、while文、do文等による繰り返し処理である。

【０１１７】

ここで、プロログ部、 エピログ部は、可能であれば、図４３（ｂ）及び（ｃ）のプロセスに示されるように、除去される。しかし、不可能であれば除去されず、コードサイズが増加することがある。そのため、ソフトウェアパイプライニング最適化の動作を指定するオプション及び#pragma指令が用意されている。

【０１１８】

図４３（ｂ）は、ループ処理におけるプロログ部・エピログ部を除去するための処理を示す概念図である。つまり、図４３（ａ）で示されたイタレーション間で依存関係のない命令X、Y、Zの５回の繰り返しについて、前記のプロログ部・カーネル部・エピログ部の概念図で示したループの生成コードイメージを並び替えたものが示されている。ただし、図中の [] のついた命令は、読み込まれるが実行されないとする。

【０１１９】

このようにすると、プロログ部・エピログ部は、カーネル部と同じ命令並びになることがわかる。よって、ループ回数は、プロログ部・エピログ部の実行分（４回）だけ増えるが、[]のついた命令をプレディケート（実行条件）によって制御することにより、図４３（ｃ）に示されるように、カーネル部だけでコードを生成することができる。

【０１２０】

図４３（ｃ）に示された生成コードの実行順序は、以下のようになる。

１回目においては、プレディケート[C2]、[C3]の付加された命令は実行されない。よって、[C4]X のみ実行される。

２回目においては、プレディケート[C2]の付加された命令は実行されない。よって、[C3]Y 、[C4]X のみ実行される。

３～５回目においては、[C2]Z、 [C3]Y、 [C4]X すべてが実行される。

６回目においては、プレディケート[C4]の付加された命令は実行されない。よって、[C2]Z と [C3]Y のみ実行される。

７回目においては、プレディケート[C3]、[C4]の付加された命令は実行されない。よって、[C2]Z のみ実行される。

【０１２１】

このように、カーネル部のループ 1回目、2回目でプロログ部を、6回目、7回目でエピログ部を実行していることになる。

【０１２２】

よって、プロログ部・エピログ部のあるループでは、コードサイズが増加するが、ループ回数が減少するため、実行速度向上を期待できる。逆に、プロログ部・エピログ部を除去したループでは、コードサイズを削減できるが、ループ回数が増加するため、実行サイクル数が増加する。

【０１２３】

そこで、このような最適化の選択を指定可能にするために、以下のコンパイルオプションとプラグマ指令が用意されている。

・コンパイルオプション

-fno-software-pipelining

・#pragma指令

#pragma _no_software_pipelining

#pragma _software_pipelining_no_proepi

#pragma _software_pipelining_with_proepi

なお、オプションとプラグマ指令が重複又は矛盾した場合には、プラグマ指令が優先する。

【０１２４】

図４４は、ソフトウェアパイプライニング部１２２の動作を示すフローチャートである。解析部１１０によってオプション「-fno-software-pipelining」が検出された場合には（ステップＳ１１０、Ｓ１１１）、ソフトウェアパイプライニング部１２２は、対象となるソースプログラム１０１中の全てのループ処理に対してソフトウェアパイプライニング最適化を行わない（ステップＳ１１２）。このオプションによって、コードサイズが増加してしまうことが回避される。

【０１２５】

また、解析部１１０によってプラグマ指令「#pragma _no_software_pipelining」が検出された場合には（ステップＳ１１０、Ｓ１１１）、ソフトウェアパイプライニング部１２２は、オプション指定にかかわらず、この指定の直後に置かれている１つのループ処理について、ソフトウェアパイプライニング最適化を行わない（ステップＳ１１３）。これによって、コードサイズが削減される。

【０１２６】

また、解析部１１０によってプラグマ指令「#pragma _software_pipelining_no_proepi」が検出された場合には（ステップＳ１１０、Ｓ１１１）、ソフトウェアパイプライニング部１２２は、オプション指定にかかわらず、この指定の直後に置かれている１つのループ処理について、プロログ部・エピログ部が除去できる範囲でソフトウェアパイプライニング最適化を行う（ステップＳ１１４）。これによって、速度の向上とサイズの削減化が図られる。

【０１２７】

また、解析部１１０によってプラグマ指令「#pragma _software_pipelining_with_proepi」が検出された場合には（ステップＳ１１０、Ｓ１１１）、ソフトウェアパイプライニング部１２２は、オプション指定にかかわらず、この指定の直後に置かれている１つのループ処理について、プロログ部・エピログ部を除去せずに、可能な範囲で、ソフトウェアパイプライニング最適化を行う（ステップＳ１１５）。これによって、速度が向上される。

【０１２８】

なお、ソフトウェアパイプライニング部１２２は、#pragma _software_pipeli

ning_no_proepi指令に対しては、プロログ部・エピログ部を除去できる範囲でソフトウェアパイプライニング最適化を行うが、#pragma _software_pipelining_with_proepi指令に対しては、プロログ部・エピログ部の除去が可能であっても、除去しない。プロログ部・エピログ部の除去可能なループであっても、図４５に示される例のように、プロログ部･エピログ部の除去を抑制することにより、コードサイズは増加するが、実行速度の向上を期待できるからである。また、後述するように、ループ処理の最低繰り返し回数がソフトウェアパイプライニングによって重なり合うイタレーション数以上である場合には、ソフトウェアパイプライニング部１２２は、ソフトウェアパイプライニングによる最適化を行う。

【０１２９】

図４５は、ソフトウェアパイプライニング最適化の例を示す図である。なお、この例では、ソフトウェアパイプライニング最適化を行うために、コンパイルオプション -O（実行速度とコードサイズ削減の最適化）をつけてコンパイルされている。

【０１３０】

本図の左欄の中下段に示された機械語プログラム１０２の例から分かるように、デフォルトのソフトウェアパイプライニング最適化が行われた場合には、プロログ部・エピログ部のコードも除去され、ループ回数が１０１回でカーネル部のサイクル数が２サイクルとなり、合計２０７サイクルで実行され、ループの性能が向上している。

【０１３１】

一方、本図の右欄の中上段に示されたソースプログラム１０１から分かるように、左欄のソースプログラム１０１に対して #pragma _software_pipelining_with_proepi指令が追加指定され、ループのプロログ部・エピログ部の除去を抑制した例となっている。これにより、右欄の中下段に示された機械語プログラム１０２の例から分かるように、プロログ部・エピログ部のコードが生成されるため、左側と比べコードサイズが増加しているが、ループ回数が９９回に減少しており、カーネル部サイクル数が２サイクルであるため、合計２０４サイクルで実行され、左欄の場合よりも更に実行速度が向上している。なお、プロログ部・エピ

ログ部が周辺コードと並列実行可能な場合には、プロログ部・エピログ部による速度低下の影響は隠蔽できる。

［ループアンローリング部１２３］

【０１３２】

次に、ループアンローリング部１２３の動作とその意義について説明する。ループアンローリング部１２３は、大きく分けて、（１）ループアンローリングの指定に関する最適化と、（２）ループの繰り返し回数についての保証に関する最適化とを行う。

【０１３３】

まず、（１）ループアンローリングの指定に関する最適化について説明する。

ループアンローリング最適化とは、ループ高速化手法の１つである。複数のイタレーションを同時に実行することでループ内の実行を高速化する。ループアンローリング最適化を行うことにより、ldp/stp命令の生成や並列度の向上により、実行速度の向上を図ることができる。しかし、コードサイズが増加することと、場合によっては、レジスタ不足によるスピルが発生し、逆に性能が低下してしまう場合がある。

【０１３４】

なお、ロードペア（ストアペア）命令（ldp/stp命令）とは、二つのロード命令（ストア命令）を１命令で実現した命令である。また、「スピル」とは、空きレジスタを確保する為に、使用されているレジスタを一時的にスタックに退避させることである。この場合には、レジスタの退避・復帰のためにロード・ストア命令が生成される。

【０１３５】

このようなループアンローリング最適化の動作を指定するオプション及び#pragma指令が用意されている。

・コンパイルオプション

-fno-loop-unroll

・#pragma指令

#pragma _loop_unroll

#pragma _no_loop_unroll

【０１３６】

図４６は、ループアンローリング部１２３の動作を示すフローチャートである。解析部１１０によってオプション「-fno-loop-unroll」が検出された場合には（ステップＳ１２０、Ｓ１２１）、ループアンローリング部１２３は、対象となるソースプログラム１０１中の全てのループ処理に対してループアンローリング最適化を行わない（ステップＳ１２２）。このオプションによって、コードサイズが増加してしまうことが回避される。

【０１３７】

また、解析部１１０によってプラグマ指令「#pragma _loop_unroll」が検出された場合には（ステップＳ１２０、Ｓ１２１）、ループアンローリング部１２３は、直後に置かれている１つのループ処理に対してループアンローリング最適化を行う（ステップＳ１２３）。これによって、速度が向上される。

【０１３８】

また、解析部１１０によってプラグマ指令「#pragma _no_loop_unroll」が検出された場合には（ステップＳ１２０、Ｓ１２１）、ループアンローリング部１２３は、直後に置かれている１つのループ処理に対してループアンローリング最適化を行わない（ステップＳ１２４）。これによってコードサイズの増加が回避される。

【０１３９】

なお、最適化レベル指定に -O/-Ot（実行速度を優先した最適化）が指定されている場合は、ループアンローリング部１２３は、ループアンローリング最適化が可能であるなら、デフォルトでループアンローリング最適化を行う。最適化レベル指定に-Os（コードサイズ削減を優先した最適化）が指定されている場合は、ループアンローリング部１２３は、ループアンローリング最適化を行わない。よって、ユーザは、これらの最適化レベル指定コンパイルオプションと組み合わせて、個々のループのループアンローリング最適化の適用を、#pragma _no_loop_unroll指令及び#pragma_loop_unroll 指令で制御することが可能である。

【０１４０】

図４７は、#pragma _loop_unroll指令による最適化の例を示す図である。本図の左欄は、最適化レベル指定コンパイルオプション -0のみをつけてコンパイルした場合の例であり、本図の右欄は、#pragma _loop_unroll指令を組み合わせてコンパイルした場合の例である。

【０１４１】

本図の左欄の中下段に示された機械語プログラム１０２の例から分かるように、プロログ部・エピログ部が除去されたソフトウェアパイプライニング最適化が適用されている。そのため、カーネル部の３命令（２サイクル）が１０１回実行され、全体として合計２０７サイクルかかっている。

【０１４２】

一方、右欄の中下段に示された機械語プログラム１０２の例から分かるように、左側と同様にソフトウェアパイプライニング最適化が行われ、プロログ部・エピログ部が削除されている。それに、この右欄の機械語プログラム１０２では、ループアンローリング最適化により、ループ回数が半減しているため、カーネル部の６命令（２サイクル）が５２回実行され、全体として合計１１０サイクルで実行され、速度が向上していいる。

【０１４３】

次に、ペアメモリアクセス命令（ldp/stp）の生成によりループアンローリング最適化をより効果的に使用する方法を示す。

【０１４４】

ループアンローリング最適化では、現在のイタレーションと次のイタレーションを同時に実行するため、以下のような連続する領域のデータのロード・ストアが生成される場合がある。

```
ld  r1,(r4);;
ld  r2,(r4,4);;
```

【０１４５】

アクセスするデータが、必ず８バイトアラインされて配置されているならば、以下のようなペアメモリアクセス命令（ldp命令）を生成することができる。

```
ldp r1:r2,(r4+);;
```

図４８は、ペアメモリアクセス命令（ldp/stp）の生成によりループアンローリング最適化をより効果的に使用する例を示す図である。ここでは、ソフトウェアパイプライニング最適化が適用されている。

【０１４６】

本図の右欄の例では、中下段に示された機械語プログラム１０２の例から分かるように、ループアンローリング最適化によりループ回数が半減されている。また、#pragma _align_local_pointer指令を使用して、ポインタ変数 pa、pbが８バイトアラインされているアドレスと明示することにより、ロードペア（ストアペア）命令が生成される。

【０１４７】

これらの最適化により、左欄の例では、カーネル部の５命令３サイクルが１０１回実行され、全体として合計３０８サイクルであるが、右欄の例では、カーネル部の７命令３サイクルが半分の５１回実行され、全体として合計１５８サイクルで実行され、速度が向上している。

【０１４８】

次に、ループアンローリング部１２３による、（２）ループの繰り返し回数の保証に関する最適化について説明する。

プログラムの記述上、コンパイラ１００ではループ回数を特定することができない場合、ループ高速化の各最適化を効果的に行うことができない。

【０１４９】

そこで、ユーザは、下記に示す#pragma指令にて、ループ回数の情報を提供することにより、より効果的にソフトウェアパイプライニング等のループ高速化の最適化を行わせることができる。

・#pragma指令

```
#pragma _min_iteration=NUM
#pragma _iteration_even
#pragma _iteration_odd
```

【０１５０】

図４６において、解析部１１０によってプラグマ指令「#pragma _min_iterati

on=NUM」が検出された場合には（ステップＳ１２０、Ｓ１２１）、ループアンローリング部１２３は、直後に置かれている１つのループ処理が最低ＮＵＭ回繰り返されることを前提に、ループアンローリング最適化を行う（ステップＳ１２５）。例えば、例えば、保証された繰り返し最低回数がループアンローリングによる展開数以上である場合に、ループアンローリング部１２３は、そのループ処理のループアンロールを行う。これによって、速度の向上とサイズの削減が図られる。

【０１５１】

また、解析部１１０によってプラグマ指令「#pragma _iteration_even」が検出された場合には（ステップＳ１２０、Ｓ１２１）、ループアンローリング部１２３は、直後に置かれている１つのループ処理が偶数回繰り返されることを前提に、ループアンローリング最適化を行う（ステップＳ１２６）。これによって、実行速度が向上される。

【０１５２】

また、解析部１１０によってプラグマ指令「#pragma _iteration_odd」が検出された場合には（ステップＳ１２０、Ｓ１２１）、ループアンローリング部１２３は、直後に置かれている１つのループ処理が奇数回繰り返されることを前提に、ループアンローリング最適化を行う（ステップＳ１２６）。これによって、実行速度が向上される。

【０１５３】

なお、#pragma _min_iteration指令で１以上の値を指定した場合、１回もループを通らない場合のために生成されるエスケープコードを除去できるという効果もある。また、繰り返し回数が不明なループに対して、ループアンローリング最適化を期待する場合、偶数回ループか奇数回ループかが決まっているならば、_iteration_even /#pragma _iteration_odd指令を使用することにより、ループアンローリング最適化の適用が可能になるため、実行速度向上を期待することができる。

【０１５４】

図４９は、#pragma _min_iteration指令による最適化の例を示す図である。こ

こでは、繰り返し回数が不明なループでの、#pragma _min_iteration指令の使用効果が示されている。ただし、サイクル比較のため、引数endの値を１００とする。

【０１５５】

本図の左欄では、中下段に示された機械語プログラム１０２の例から分かるように、ループ回数が不明なため、一度もループを実行しない場合にループ本体を飛び越すための cmple/br 命令（エスケープコード）が生成されている。また、ループ命令の生成を行うことができないため、加算命令・比較命令・ジャンプ命令でループが生成されている。サイクル数は、ループ部が７命令４サイクルの１００回繰り返しとなり、全体として合計４０５サイクルとなっている。

【０１５６】

一方、本図の右欄では、中上段に示されたソースプログラム１０１の例から分かるように、繰り返し回数が不明であるが、最低４回繰り返されることが#pragma _min_iteration指令で指定されている。これにより、ループ回数が０回の場合を考慮する必要が無いため、ループアンローリング部１２３は、エスケープコードを生成する必要がなくなる。

【０１５７】

また、ループ最低回数を考慮して、ループアンローリング部１２３は、ループ命令を生成することができる。例えば、保証された繰り返し最低回数（４）がループアンローリングによる展開数（この例では、３サイクル）以上であるので、ループアンローリング部１２３は、ループアンロールを行う。

【０１５８】

さらに、この例では、さらにソフトウェアパイプライニング最適化が可能になっている。これは、保証されたループの繰り返し最低回数（４）がソフトウェアパイプライニングによって重なり合うイタレーション数以上であったために、ソフトウェアパイプライニング部１２２がソフトウェアパイプライニングによる最適化を行ったためである。

【０１５９】

右欄の中下段に示された機械語プログラム１０２の例から分かるように、サイ

クル数は、ループ部が５命令３サイクルの１０１回繰り返しとなり、全体として合計３０８サイクルとなり、実行速度とサイズ削減が実現されている。

【０１６０】

図５０及び図５１は、#pragma iteration_even／#pragma _iteration_odd 指令による最適化の例を示す図である。図５０は、ループ回数が不明の場合におけるソースプログラム１０１の例（左欄）と、そこから生成される機械語プログラム１０２の例（右欄）を示す図である。本図から分かるように、実際のループ回数が不明な場合、ループアンローリング最適化は適用できない。これは、ループ回数が偶数回の場合と奇数回の場合では、ループアンローリング最適化によって生成されるコードが異なるためである。

【０１６１】

ところが、図５１に示されるように、繰り返し回数が不明なループの場合でも、遇数回ループか奇数回ループかを指定することにより、ループアンローリング最適化を適用することができる。

【０１６２】

本図の左欄では、ループ回数が偶数回であることを、#pragma _iteration_even指令で指定されているため、ループアンローリング部１２３によるループアンローリング最適化が行われ、左欄の中下段に示された機械語プログラム１０２の例から分かるように、偶数回用のコードが生成されている。

【０１６３】

また、本図の右欄では、ループ回数が奇数回であることを、#pragma _iteration_odd指令により指定されているため、ループアンローリング部１２３によるループアンローリング最適化が行われ、右欄の中下段に示された機械語プログラム１０２の例から分かるように、奇数回用のコードが生成されている。この右欄の例から分かるように、左欄に示された偶数回の場合の生成コードと初期化部・ループ部はほぼ同じで、後処理部に、ループの最後の一回分を実行するコードが生成されている。

【０１６４】

このように、ループ回数が不明であっても、偶数回であるか奇数回であるかを

保証することで、ループアンローリング部１２３は、ループアンローリング最適化を行うことができ、これによって実行速度が向上される。

【０１６５】

［ｉｆ変換部１２４］

次に、ｉｆ変換部１２４の動作とその意義について説明する。

通常、Ｃ言語プログラムのｉｆ構造をコンパイルすると、分岐命令（br命令）が生成される。これに対して、ｉｆ変換とは、Ｃ言語プログラムのｉｆ構造を分岐命令を用いることなく、条件付き実行命令だけに書き換えることである。これによって、実行順序が固定化される（順次実行となる）るので、パイプラインの乱れが回避され、実行速度が向上され得る。なお、条件付き実行命令とは、その命令に含まれる条件（プレディケート）がプロセッサ１の状態（コンディションフラグ）と一致している場合にだけ実行される命令である。

【０１６６】

ｉｆ変換により、ｉｆ構造のワーストケースにおける実行時間は短縮されるが、ベストケースにおける実行時間は（短縮後の）ワースト実行時間と等しくなる。そのために、ｉｆ構造の特性（条件成立・不成立それぞれの発生頻度や各パスの実行サイクル数）に応じて、ｉｆ変換を適用すべき場合とすべきでない場合がある。

【０１６７】

このため、ユーザは、適用の可否をコンパイルオプションや#pragma指令で指示することができる。

・コンパイルオプション

-fno-if-conversion

・#pragma指令

#pragma _if_conversion

#pragma _no_if_conversion

なお、オプションとプラグマ指令が重複又は矛盾した場合には、プラグマ指令が優先する。

【０１６８】

図５２は、ｉｆ変換部１２４の動作を示すフローチャートである。解析部１１０によってオプション「-fno-if-conversion」が検出された場合には（ステップＳ１３０、Ｓ１３１）、ｉｆ変換部１２４は、対象となるソースプログラム１０１中の全てのｉｆ構造文に対してｉｆ変換を行わない（ステップＳ１３２）。なお、本オプションが検出されない場合は、ｉｆ変換部１２４は、ｉｆ変換が可能であり、かつ、そのワーストケースの時間がｉｆ変換前に対して短いｉｆ構造文である場合に、そのｉｆ構造文をｉｆ変換する。

【０１６９】

また、解析部１１０によってプラグマ指令「#pragma _if_conversion」が検出された場合には（ステップＳ１３０、Ｓ１３１）、ｉｆ変換部１２４は、オプション指定にかかわらず、直後に置かれている１つのｉｆ構造文に対して、可能であればｉｆ変換を行う（ステップＳ１３３）。これによって、速度が向上される。

【０１７０】

また、解析部１１０によってプラグマ指令「#pragma _no_if_conversion」が検出された場合には（ステップＳ１３０、Ｓ１３１）、ｉｆ変換部１２４は、オプション指定にかかわらず、直後に置かれている１つのｉｆ構造文に対して、ｉｆ変換を行わない（ステップＳ１３４）。これによって、速度が向上される。

【０１７１】

図５３は、#pragma _no_if_conversion指令でコンパイルした場合と、#pragma _if_conversion指令でコンパイルした場合の機械語プログラム１０２の例を示す図である。

【０１７２】

本図の左欄では、中下段の機械語プログラム１０２の例から分かるように、ｉｆ変換を抑制したことにより、分岐命令が生成されている（実行サイクル数：５あるいは７、 コードサイズ：１２バイト）。

【０１７３】

一方、本図の右欄では、中下段の機械語プログラム１０２の例から分かるように、#pragma指令によってｉｆ変換を行うこととしたことにより、分岐命令が、

条件付き命令（プレディケート付き命令）に置き換わっている（実行サイクル数：４、コードサイズ：８バイト）。このように、ｉｆ変換を実施することで、実行速度比１．２５倍、コードサイズ比６７％が達成されている。

【０１７４】

［ペア命令生成部１２５］

次に、ペア命令生成部１２５の動作とその意義について説明する。ペア命令生成部１２５は、大きく分けて、（１）配列・構造体のアラインメントの設定に関する最適化と、（２）仮引数ポインタ・ローカルポインタのアラインの保証に関する最適化とを行う。

【０１７５】

まず、（１）配列・構造体のアラインメントの設定に関する最適化について説明する。

ユーザは、以下のオプションを用いて、配列と構造体の先頭アドレスのアラインを指定することができる。アラインメントを調整することで、メモリアクセス命令のペアリング（２つのレジスタとメモリ間の転送をひとつの命令で行うこと）が可能となり、実行速度の向上が期待できる。その反面、アラインメント値を大きくすると、データの未使用領域が増加し、データサイズが増大する可能性がある。

・コンパイルオプション

-falign_char_array=NUM　(NUM=2,4または8)

-falign_short_array=NUM　(NUM=4または8)

-falign_int_array=NUM　(NUM=8)

-falign_all_array=NUM　(NUM=2,4または8)

-falign_struct=NUM　(NUM=2,4または8)

【０１７６】

上記オプションは、上から順に、char型の配列、short型整数、int型整数、それら３つののデータ型全ての配列、構造体のアラインメントを指定している。また、"NUM"は、アラインするサイズ（バイト）を示す。

【０１７７】

図５４は、ペア命令生成部１２５の動作を示すフローチャートである。解析部１１０によって上記オプションのいずれかが検出された場合には（ステップＳ１４０、Ｓ１４１）、ペア命令生成部１２５は、対象となるソースプログラム１０１で宣言されている指定された型の全ての配列又は構造体について、その先頭アドレスが指定されたＮＵＭバイトのアラインとなるように配列又は構造体をメモリに配置し、その配列又は構造体にアクセスする命令については、可能な場合に、ペアリング（２つのレジスタとメモリ間の転送を並行して行う命令の生成）を行う（ステップＳ１４２）。これによって、実行速度が向上される。

【０１７８】

図５５は、サンプルプログラムをオプションなしでコンパイルした場合と、オプション'-falign-short-array=4'でコンパイルした場合のアセンブリコードを示す図である。

【０１７９】

本図の左欄に示されたオプションなしの場合、中下段に示された機械語プログラム１０２の例から分かるように、アラインメントが不明のため、ロード命令のペアリング（２つのレジスタとメモリ間の転送をひとつの命令で行う）ができない（実行サイクル数：２５、 コードサイズ：２２）。

【０１８０】

一方、本図の右欄に示されたオプションありの場合、配列が４バイトでアラインされるため、中下段に示された機械語プログラム１０２の例から分かるように、最適化部１２０によるペアリングが実現されている（実行サイクル数：１５、コードサイズ：１８）。このように、アラインメントの指定によって、実行速度比１．６７倍、 コードサイズ比８２％が達成されている。

【０１８１】

次に、ペア命令生成部１２５による、（２）仮引数ポインタ・ローカルポインタのアラインの保証に関する最適化を説明する。

【０１８２】

ユーザは、以下のプラグマ指令を用いて、関数引数のポインタ変数の指すデータのアラインメントや、ローカルポインタ変数の指すデータのアラインメントを

保証することで、最適化部１２０によるメモリアクセス命令のペアリングが可能となり、実行速度の向上が期待できる。

・#pragma指令

#pragma _align_parm_pointer=NUM 変数名［，変数名，…］

#pragma _align_local_pointer=NUM 変数名［，変数名，…］

【０１８３】

なお、"NUM"はアラインするサイズ（２，４又は８バイト）を表す。また、上記#pragma指令で保証されたポインタ変数の指すデータが指定されたバイト境界にアラインされていなかった場合には、プログラムの正常動作は保証されない。

【０１８４】

図５４において、解析部１１０によってプラグマ指令「#pragma _align_parm_pointer=NUM 変数名［，変数名，…］」が検出された場合には（ステップＳ１４０、Ｓ１４１）、ペア命令生成部１２５は、"変数名"で示される引数のポインタ変数の指すデータが引数渡しの時点でＮＵＭバイトにアラインされているものとし、その配列にアクセスする命令については、可能な場合に、ペアリングを行う（ステップＳ１４３）。これによって、実行速度が向上される。

【０１８５】

また、解析部１１０によってプラグマ指令「#pragma _align_local_pointer=NUM 変数名［，変数名，…］」が検出された場合には（ステップＳ１４０、Ｓ１４１）、ペア命令生成部１２５は、"変数名"で示されるローカルポインタ変数の指すデータが関数内部で常にNUMバイトでアラインされているものとし、その配列にアクセスする命令については、可能な場合に、ペアリングを行う（ステップＳ１４４）。これによって、実行速度が向上される。

【０１８６】

図５６は、プラグマ指令「#pragma _align_parm_pointer=NUM 変数名［，変数名，…］」による最適化の例を示す図である。

【０１８７】

本図の左欄に示されるように、#pragma _align_parm_pointerを与えない場合、ポインタ変数srcの指すデータのアラインメントが不明なため、中下段に示さ

れた機械語プログラム１０２の例から分かるように、各データはそれぞれ独立にロードされる（実行サイクル数１６０、コードサイズ：２４バイト）。

【０１８８】

一方、本図の右欄に示されるように、#pragma指令を与えると、データは４バイト境界にアラインされるため、中下段の機械語プログラム１０２の例から分かるように、メモリ読み出しのペアリングが行われる（実行サイクル数：１０７、コードサイズ：１８バイト）。このように、アライメントを指定することで、実行速度比１．５０倍、コードサイズ比４３％が達成される。

【０１８９】

図５７は、プラグマ指令「#pragma _align_local_pointer=NUM 変数名［，変数名，…］」による最適化の例を示す図である。

本図の左欄に示されるように、#pragma _align_local_pointerを与えない場合、ポインタ変数from、toの指すデータのアラインメントが不明なため、中下段に示された機械語プログラム１０２の例から分かるように、配列要素はそれぞれ独立にロードされる（実行サイクル数：７２、コードサイズ：３０）。

【０１９０】

一方、本図の右欄に示されるように、#pragma _align_parm_pointerを与えることで、中下段に示された機械語プログラム１０２の例から分かるように、ポインタ変数from、toの指すデータが４バイト境界にアラインされていることを利用したメモリ読み出しのペアリングが可能となる。（実行サイクル数：５６、コードサイズ：２２）。このように、アライメントを指定することで、実行速度比１．３２倍、コードサイズ比７３％が達成される。

【０１９１】

【発明の効果】

以上の説明から明らかように、本発明に係るコンパイラは、変数のグローバル領域への割り付けに関する指示を受け付け、その指示に基づいて、各種変数のグローバル領域へのマッピングを行う。

【０１９２】

その１つとして、ユーザは、コンパイル時のオプションによって、グローバル

領域に割り付ける変数の最大データサイズを指定することができる。これによって、ユーザは、グローバル領域に配置させる変数のデータサイズを制御することが可能となり、グローバル領域を有効活用するように最適化を図ることができる。

【０１９３】

また、ユーザは、ソースプログラム中に置くプラグマ指令によって、変数ごとに、グローバル領域への割り付けをする／しない旨の指定をすることができる。これによって、グローバル領域に優先的に割り付けるべき変数、割り付けてはいけない変数を個々に区別して、最適なグローバル領域の割付けをユーザが管理することができる。

【０１９４】

また、本発明に係るコンパイラは、ソフトウェアパイプライニングの指示を受け付け、その指示に従ったソフトウェアパイプライニングによる最適化を行う。

【０１９５】

その１つとして、ユーザは、コンパイル時のオプションによって、ソフトウェアパイプライニングをしない旨の指定をすることができる。これによって、ソフトウェアパイプライニングによるコードサイズの増加が抑制される。またソフトウェアパイプライニングが行われたアセンブラコードは複雑なので、プログラムの機能検証のために、ソフトウェアパイプライニングを抑制するこで、デバッグが容易となる。

【０１９６】

また、ユーザは、ソースプログラム中のプラグマ指令によって、ループ処理ごとに、ソフトウェアパイプライニングをする／しない旨の指定をしたり、プロログ部及びエピログ部を除去して／除去しないでソフトウェアパイプライニングをする旨の指定をしたりすることができる。これによって、ループ処理ごとにソフトウェアパイプライニングをするか否かを選択したり、コードサイズ重視(プロローグエピローグ除去する)又はスピード重視(プロローグエピローグ除去しない)のソフトウェアパイプライニングを選択したりすることが可能となる。

【０１９７】

また、本発明に係るコンパイラは、ループアンローリングの指示を受け付け、その指示に従ったループアンローリングによる最適化を行う。

【０１９８】

その１つとして、ユーザは、コンパイル時のオプションによって、ループアンローリングをしない旨の指示をすることができる。これによって、ループアンローリングによるコードサイズの増加が回避される。

【０１９９】

また、ユーザは、ソースプログラム中のプラグマ指令によって、ループ処理ごとに、ループアンローリングをする／しない旨の指示をすることができる。これによって、ユーザは、ループ処理ごとに、その繰り返し回数等を勘案し、実行速度を重視するか、コードサイズを重視するかの最適化を選択することができる。

【０２００】

また、本発明に係るコンパイラは、ループ処理の繰り返し回数に関する指示を受け付け、その指示に従った最適化を行う。

【０２０１】

その１つとして、ユーザは、ソースプログラム中のプラグマ指令によって、ループ処理ごとに、最低の繰り返し回数を保証することができる。これによって、繰り返し回数が０である場合に必要とされるコード（エスケープコード）を生成する必要がなくなるとともに、ソフトウェアパイプライニングやループアンロールによる最適化が可能となる。

【０２０２】

また、ユーザは、ソースプログラム中のプラグマ指令によって、ループ処理ごとに、繰り返し回数が偶数回／奇数回であることを保証することができる。これによって、たとえ繰り返し回数が不明であっても、ループ処理ごとに、ループアンロールによる最適化が可能となり、実行速度が向上され得る。

【０２０３】

また、本発明に係るコンパイラは、ｉｆ変換に関する指示を受け付け、その指示に従ったｉｆ変換による最適化を行う。

【０２０４】

その１つとして、ユーザは、コンパイル時のオプションによって、ｉｆ変換をしない旨の指示をすることができる。これによって、ｉｆ構造のthen側とelse側で命令数のバランスが悪い場合に、ｉｆ変換によって命令数の少ない側の実行が命令数の多い側に制約されてしまうという不具合の発生を防ぐことができる。

【０２０５】

また、ユーザは、ソースプログラム中のプラグマ指令によって、ループ処理ごとに、ｉｆ変換をする／しない旨の指示をすることができる。これによって、個々のループ処理の特性（then側とelse側それぞれの命令数のバランス、予想される発生頻度のバランス等）を考慮して、より実行速度が向上されると予測される選択（ｉｆ変換をする／しない）を行うことができる。

【０２０６】

また、本発明に係るコンパイラは、配列データのメモリ領域への配置におけるアライメントに関する指示を受け付け、その指示に従った最適化を行う。

【０２０７】

その１つとして、ユーザは、コンパイル時のオプションによって、特定の型の配列データについて、バイト数によるアラインを指定することができる。これによって、２つのデータに対するメモリ・レジスタ間の転送を同時に行うペア命令が生成され、実行速度が向上される。

【０２０８】

また、ユーザは、ソースプログラム中のプラグマ指令によって、ポインタ変数が指すデータのアラインを指定することができる。これによって、データごとに、ペア命令が生成されることを可能にすることができ、実行速度が向上される。

【０２０９】

以上のように、本発明に係るコンパイラにより、ユーザは、従来のような大雑把な指示ではなく、コンパイラによる各種最適化の種類ごとにＯＮ／ＯＦＦやその程度を指定したり、プログラム中の変数やループ処理等の単位で最適化をＯＮ／ＯＦＦさせたりする等のきめ細かい制御が可能となり、特に、精密な最適化のチューニングが必要とされるメディア処理のアプリケーション開発に有効であり、その実用的価値は極めて高い。

【図面の簡単な説明】

【図１】　本発明に係るコンパイラの対象となるプロセッサの概略ブロック図である。

【図２】　同プロセッサの算術論理・比較演算器の概略図を示す。

【図３】　同プロセッサのバレルシタの構成を示すブロック図である。

【図４】　同プロセッサの変換器の構成を示すブロック図である。

【図５】　同プロセッサの除算器の構成を示すブロック図である。

【図６】　同プロセッサの乗算・積和演算器の構成を示すブロック図である。

【図７】　同プロセッサの命令制御部の構成を示すブロック図である。

【図８】　同プロセッサの汎用レジスタ（Ｒ０～Ｒ３１）の構造を示す図である。

【図９】　同プロセッサのリンクレジスタ（ＬＲ）の構造を示す図である。

【図１０】　同プロセッサの分岐レジスタ（ＴＡＲ）の構造を示す図である。

【図１１】　同プロセッサのプログラム状態レジスタ（ＰＳＲ）の構造を示す図である。

【図１２】　同プロセッサの条件フラグレジスタ（ＣＦＲ）の構造を示す図である。

【図１３】　同プロセッサのアキュムレータ（Ｍ０，Ｍ１）の構造を示す図である。

【図１４】　同プロセッサのプログラムカウンタ（ＰＣ）の構造を示す図である。

【図１５】　同プロセッサのＰＣ退避用レジスタ（ＩＰＣ）の構造を示す図である。

【図１６】　同プロセッサのＰＳＲ退避用レジスタ（ＩＰＳＲ）の構造を示す図である。

【図１７】　同プロセッサのパイプライン動作を示すタイミング図である。

【図１８】　同プロセッサによる命令実行時の各パイプライン動作を示すタ

イミング図である。

【図１９】　同プロセッサの並列動作を示す図である。

【図２０】　同プロセッサが実行する命令のフォーマットを示す図である。

【図２１】　カテゴリー「ALUadd（加算）系」に属する命令を説明する図である。

【図２２】　カテゴリー「ALUsub（減算）系」に属する命令を説明する図である。

【図２３】　カテゴリー「ALUlogic（論理演算）系ほか」に属する命令を説明する図である。

【図２４】　カテゴリー「CMP（比較演算）系」に属する命令を説明する図である。

【図２５】　カテゴリー「mul（乗算）系」に属する命令を説明する図である。

【図２６】　カテゴリー「mac（積和演算）系」に属する命令を説明する図である。

【図２７】　カテゴリー「msu（積差演算）系」に属する命令を説明する図である。

【図２８】　カテゴリー「MEM ld（メモリ読み出し）系」に属する命令を説明する図である。

【図２９】　カテゴリー「MEMstore（メモリ書き出し）系」に属する命令を説明する図である。

【図３０】　カテゴリー「BRA（分岐）系」に属する命令を説明する図である。

【図３１】　カテゴリー「BSasl（算術バレルシフト）系ほか」に属する命令を説明する図である。

【図３２】　カテゴリー「BSlsr（論理バレルシフト）系ほか」に属する命令を説明する図である。

【図３３】　カテゴリー「CNVvaln（算術変換）系」に属する命令を説明する図である。

【図３４】　カテゴリー「CNV（一般変換）系」に属する命令を説明する図である。

【図３５】　カテゴリー「SATvlpk（飽和処理）系」に属する命令を説明する図である。

【図３６】　カテゴリー「ETC（その他）系」に属する命令を説明する図である。

【図３７】

本発明に係るコンパイラの構成を示す機能ブロック図である。

【図３８】

（ａ）は、グローバル領域におけるデータ等の配置例、（ｂ）は、グローバル領域以外の領域におけるデータの配置例を示す図である。

【図３９】

グローバル領域割り付け部の動作を示すフローチャートである。

【図４０】

グローバル領域割り付け部による最適化の具体例を示す図である。

【図４１】

グローバル領域割り付け部による最適化の具体例を示す図である。

【図４２】

グローバル領域割り付け部による最適化の具体例を示す図である。

【図４３】

ソフトウェアパイプライニング最適化を説明する図である。

【図４４】

ソフトウェアパイプライニング部の動作を示すフローチャートである。

【図４５】

ソフトウェアパイプライニング部による最適化の具体例を示す図である。

【図４６】

ループアンローリング部の動作を示すフローチャートである。

【図４７】

ループアンローリング部による最適化の具体例を示す図である。

【図４８】

ループアンローリング部による最適化の具体例を示す図である。

【図４９】

ループアンローリング部による最適化の具体例を示す図である。

【図５０】

ループアンローリング最適化が適用できない例を示す図である。

【図５１】

ループアンローリング部による最適化の具体例を示す図である。

【図５２】

ｉｆ変換部の動作を示すフローチャートである。

【図５３】

ｉｆ変換部による最適化の具体例を示す図である。

【図５４】

ペア命令生成部の動作を示すフローチャートである。

【図５５】

ペア命令生成部による最適化の具体例を示す図である。

【図５６】

ペア命令生成部による最適化の具体例を示す図である。

【図５７】

ペア命令生成部による最適化の具体例を示す図である。

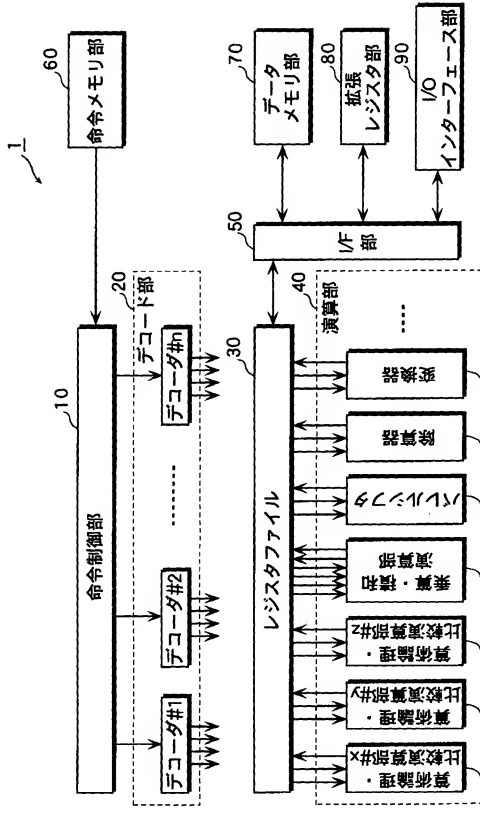
【符号の説明】

１	プロセッサ
１０	命令制御部
２０	デコード部
３０	レジスタファイル
３１	プログラム状態レジスタ（ＰＳＲ）
３２	条件フラグレジスタ（ＣＦＲ）
３３	プログラムカウンタ（ＰＣ）
３４	ＰＣ退避用レジスタ（ＩＰＣ）

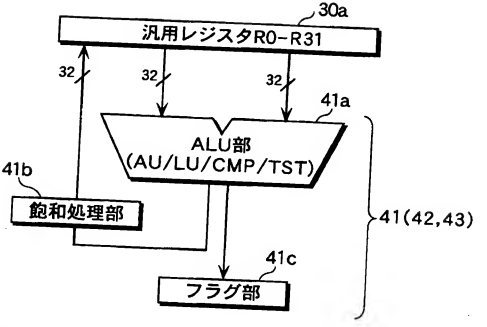
３５　ＰＳＲ退避用レジスタ（ＩＰＳＲ）
４０　演算部
４１～４３　算術論理・比較演算器
４４　積和演算器
４５　バレルシフタ
４６　除算器
４７　変換器
５０　Ｉ／Ｆ部
６０　命令メモリ部
７０　データメモリ部
８０　拡張レジスタ部
９０　Ｉ／Ｏインターフェース部
１００　コンパイラ
１０１　ソースプログラム
１０２　機械語プログラム
１１０　解析部
１１２　ステップＳ
１２０　最適化部
１２１　グローバル領域割り付け部
１２２　ソフトウェアパイプライニング部
１２３　ループアンローリング部
１２４　ｉｆ変換部
１２５　ペア命令生成部
１３０　出力部

【書類名】　図面

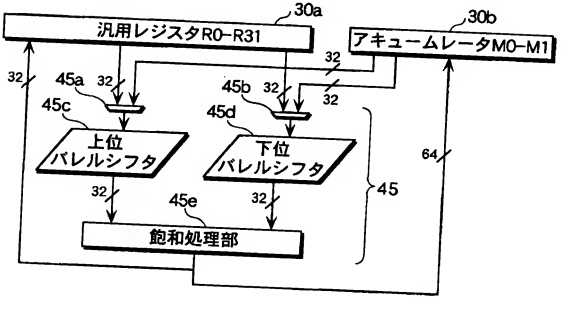
【図１】


1
10
命令制御部
20
デコード部
デコーダ#1
デコーダ#2
デコーダ#n
30
レジスタファイル
40
演算部
41
算術論理・比較演算部#X
42
算術論理・比較演算部#Y
43
算術論理・比較演算部#Z
44
乗算・積和演算部
45
シフト
46
除算器
47
変換器
50
I/F部
60
命令メモリ部
70
データメモリ部
80
拡張レジスタ部
90
I/Oインターフェース部

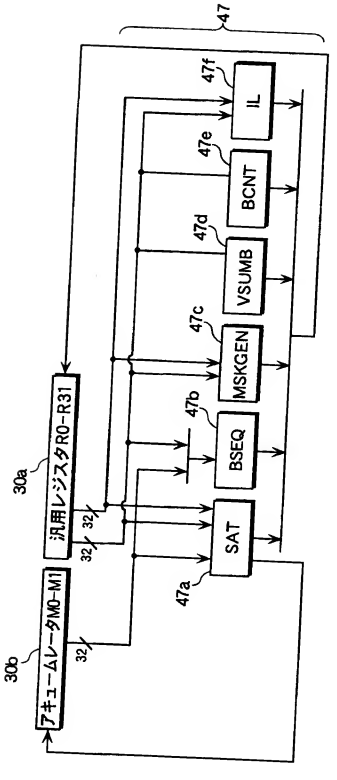
【図２】


30a
汎用レジスタR0-R31
32
32
32
41a
ALU部
(AU/LU/CMP/TST)
41b
飽和処理部
41(42,43)
41c
フラグ部

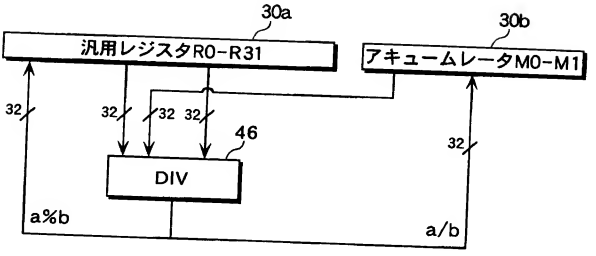
【図３】


30a
汎用レジスタR0-R31
30b
アキュームレータM0-M1
32
32
45a
32
45b
32
32
45c
45d
上位
バレルシフタ
下位
バレルシフタ
64
45
32
32
45e
飽和処理部

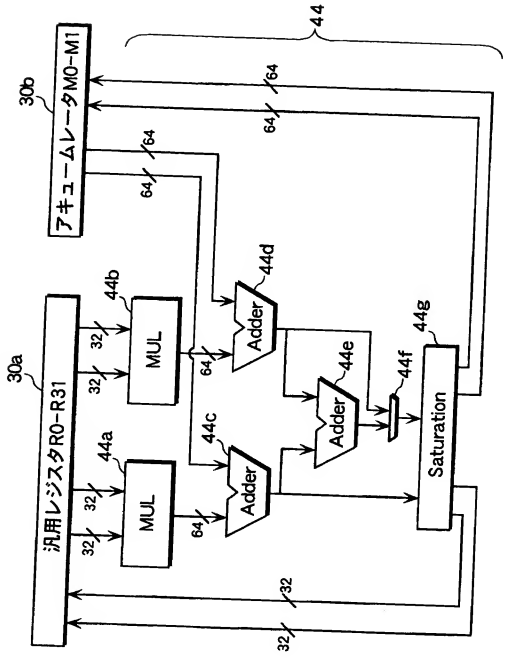
【図４】


47
47f
IL
47e
BCNT
47d
VSUMB
47c
MSKGEN
47b
BSEQ
SAT
47a
30a
汎用レジスタR0-R31
32
32
30b
アキュームレータM0-M1
32

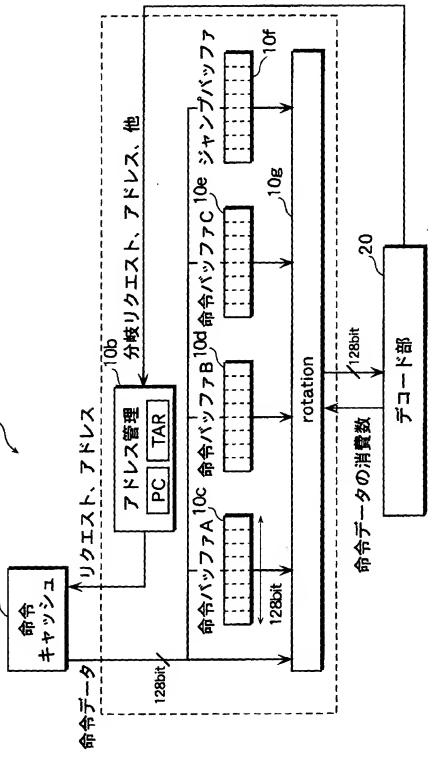
【図５】



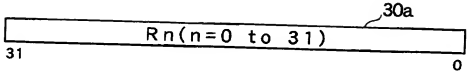
【図６】


44
30b
アキュムレータM0-M1
64
64
64
64
30a
汎用レジスタR0-R31
32
32
32
32
32
32
44b
MUL
44a
MUL
44d
Adder
44c
Adder
44e
Adder
44f
44g
Saturation
64
64

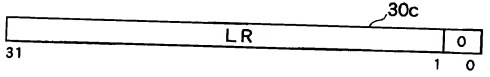
【図７】


10
10a
命令
キャッシュ
リクエスト、アドレス
命令データ
128bit
10b
分岐リクエスト、アドレス、他
アドレス管理
PC
TAR
命令バッファA
10c
命令バッファB
10d
命令バッファC
10e
ジャンプバッファ
10f
128bit
rotation
10g
命令データの消費数
128bit
20
デコード部

【図８】

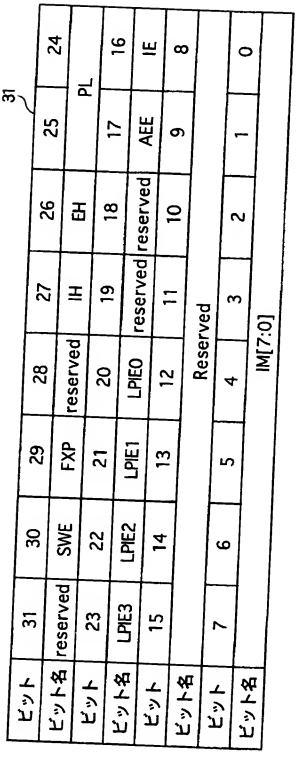


【図９】

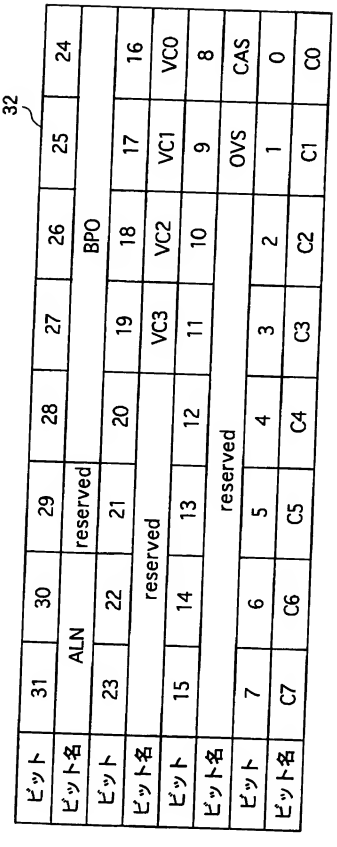


【図１０】

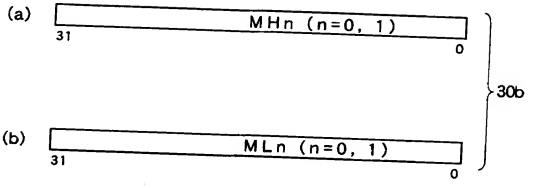
【図１１】


31
ビット 31 30 29 28 27 26 25 24
ビット名 reserved SWE FXP reserved IH EH PL
ビット 23 22 21 20 19 18 17 16
ビット名 LPIE3 LPIE2 LPIE1 LPIE0 reserved reserved AEE IE
ビット 15 14 13 12 11 10 9 8
ビット名 Reserved
ビット 7 6 5 4 3 2 1 0
ビット名 IM[7:0]

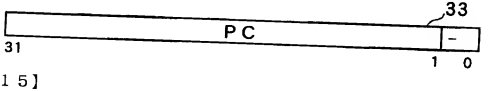
【図１２】


32
ビット 31 30 29 28 27 26 25 24
ビット名 ALN reserved BPO
ビット 23 22 21 20 19 18 17 16
ビット名 reserved VC3 VC2 VC1 VC0
ビット 15 14 13 12 11 10 9 8
ビット名 reserved OVS CAS
ビット 7 6 5 4 3 2 1 0
ビット名 C7 C6 C5 C4 C3 C2 C1 C0

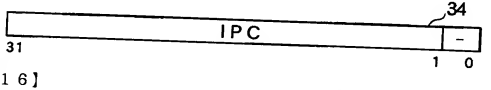
【図１３】



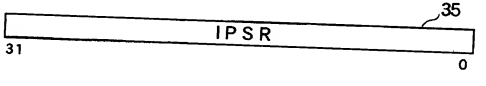
【図１４】



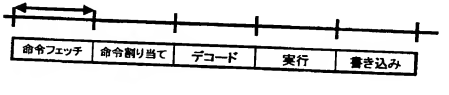
【図１５】



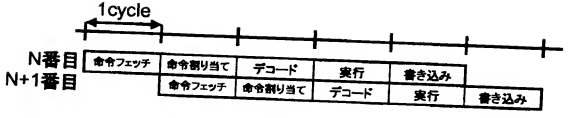
【図１６】



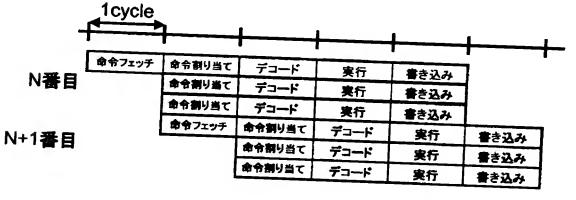
【図１７】



【図１８】

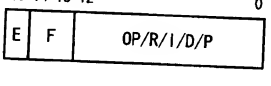


【図１９】

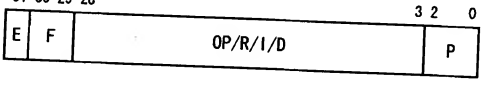


【図２０】

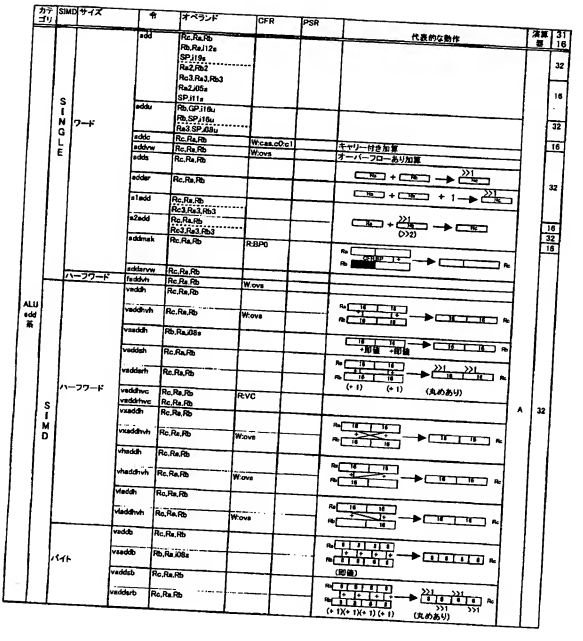
（ａ）１６ビット命令フォーマット



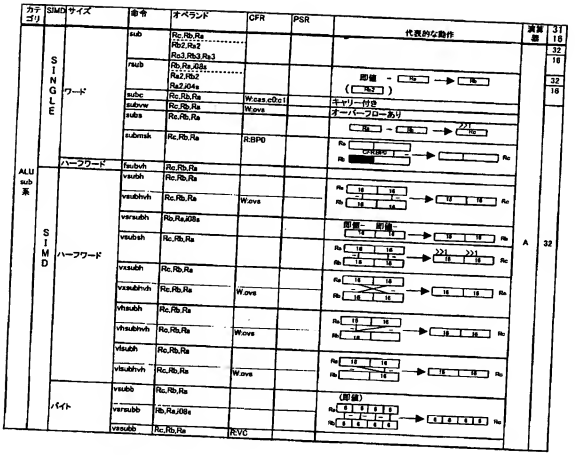
（ｂ）３２ビット命令フォーマット



【図２１】


カテゴリ
SIMD
サイズ
命令
オペランド
CFR
PSR
代表的な動作
演算器
31
16
32
ALU add 系
SINGLE
ワード
add
Rc,Ra,Rb
Rb,Ra,I12s
SP,I16u
Ra2,Rb2
Ra3,Rb3
Ra2,I05u
SP,I11s
addu
Rb,GP,I16u
Rb,SP,I16u
Ra3,SP,I08u
addc
Rc,Ra,Rb
W:cas,cO:cI
キャリー付き加算
addvw
Rc,Ra,Rb
W:ovs
オーバーフロー有り加算
subs
Rc,Ra,Rb
adder
Rc,Ra,Rb
s1add
Rc,Ra,Rb
Rc3,Ra3,Rb3
s2add
Rc,Ra,Rb
Rc3,Ra3,Rb3
addmsk
Rc,Ra,Rb
R:SPR
ハーフワード
addsrvw
Rc,Ra,Rb
SIMD
ハーフワード
fxddvh
Rc,Ra,Rb
W:ovs
vaddh
Rc,Ra,Rb
vaddhvh
Rc,Ra,Rb
W:ovs
vsaddh
Rb,Ra,I08s
+即値
vadddh
Rc,Ra,Rb
vaddsrh
Rc,Ra,Rb
(+ 1)
(+ 1)
(丸めあり)
vaddhvc
Rc,Ra,Rb
R:VC
vadddhvc
Rc,Ra,Rb
vsaddh
Rc,Ra,Rb
vxaddhvh
Rc,Ra,Rb
W:ovs
vhaddh
Rc,Ra,Rb
vhaddhvh
Rc,Ra,Rb
W:ovs
vfaddh
Rc,Ra,Rb
vfaddhvh
Rc,Ra,Rb
W:ovs
バイト
vaddb
Rc,Ra,Rb
vsaddb
Rb,Ra,I08s
(即値)
vadddb
Rc,Ra,Rb
vaddsrb
Rc,Ra,Rb
(+ 1)(+ 1)(+ 1)(+ 1)
(丸めあり)
A
32
32
16
32
16
32
16
32
16
32
16
32

【図２２】


カテゴリ
SIMD
サイズ
命令
オペランド
CFR
PSR
代表的な動作
演算器
31
16
32
16
32
16
ALU sub系
SINGLE
ワード
sub
Rc,Rb,Ra
Ra2,Ra2
Ra3,Rb3,Ra3
rsub
Rb,Ra,d8s
Ra3,Rb3
Ra2,d04s
即値
（符号）
subc
Rc,Rb,Ra
W:cas,c0o1
キャリー付き
subvw
Rc,Rb,Ra
W:ovs
オーバーフローあり
suba
Rc,Rb,Ra
submsk
Rc,Rb,Ra
R:SPD
CPUSPD
ハーフワード
fsubvh
Rc,Rb,Ra
SIMD
ハーフワード
vsubh
Rc,Rb,Ra
vsubhvh
Rc,Rb,Ra
W:ovs
vsrsubh
Rb,Ra,d08s
vssubh
Rc,Rb,Ra
即値
vxsubh
Rc,Rb,Ra
vxsubhvh
Rc,Rb,Ra
W:ovs
vhsubh
Rc,Rb,Ra
vhsubhvh
Rc,Rb,Ra
W:ovs
vlsubh
Rc,Rb,Ra
vlsubhvh
Rc,Rb,Ra
W:ovs
バイト
vsubb
Rc,Rb,Ra
vsrsubb
Rb,Ra,d08s
vssubb
Rc,Rb,Ra
R:VC
(即値)
A
32

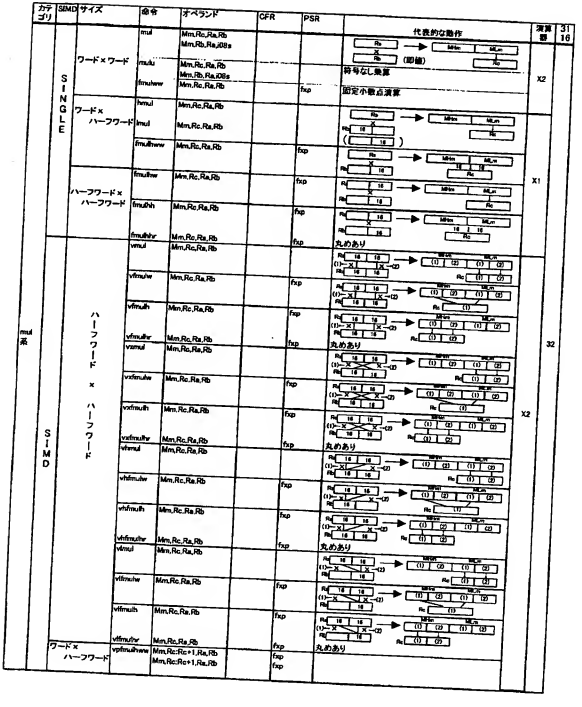
【図２３】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 / 16
ALU logic系	SINGLE	ワード	and	Rc,Ra,Rb Rb,Ra,IOBu Ra2,Rb2			論理積	A	32 16 32 16 32 16
			andn	Rc,Ra,Rb Rb,Ra,IOBu Ra2,Rb2					
			or	Rc,Ra,Rb Rb,Ra,IOBu Ra2,Rb2			論理和		
			xor	Rc,Ra,Rb Rb,Ra,IOBu Ra2,Rb2			排他的論理和		
ALU mov系	SINGLE	ワード	mov	Rb,Reg32 Reg32,Rb Rb2,Reg16 Reg16,Rb2 Ra2,Rb Ra,IIBs Ra2,IOBs			Reg32 = TAR LR SVR PSR CFR MH0 MH1 ML0 ML1 EPSR IPC PSR PC EPC PSR0 PSR1 PSR2 PSR3 CFR0 CFR1 CFR2 CFR3 Reg16 = TAR LR MH0 MH1	A	32 16 32 16
			movp	Rc:Rc+1,Ra,Rb			Rc<-Ra; Rc+1<-Rb;		32
			movcf	Ck,Cj,Cm,Cn			Cj<-Ck; Cm<-Cn;		
			mvcbove	Cm:Cn+1	W:ove		Cm:Cn+1<-CFR.OVS^CFR.OVS; CFR.OVS<-0;		16
			mvcbcas	Cm:Cn+1	W:cas		Cm:Cn+1<-CFR.CAS^CFR.CAS; CFR.CAS<-0;		
			set2d	Ra,IIBs					32
ALU max min系	SIMD	ワード	max	Rc,Ra,Rb	W:c0,c1		Rc <- max(Ra,Rb)	A	32
			min	Rc,Ra,Rb	W:c0,c1		Rc <- min(Ra,Rb)		
		ハーフワード	vmaxh	Rc,Ra,Rb					
			vminh	Rc,Ra,Rb					
		バイト	vmaxb	Rc,Ra,Rb					
			vminb	Rc,Ra,Rb					
ALU abs系	SIMD		abs	Rb,Ra			絶対値	A	32
	GLE		absvw	Rb,Ra	W:ove		オーバーフロー有り		
		ハーフワード	absvh	Rb,Ra	W:ove				
	SIMD		vabshvh	Rb,Ra	W:ove				
ALU neg系	SIMD	ワード	negvw	Rb,Ra	W:ove			A	32
	GLE	ハーフワード	fnegvh	Rb,Ra	W:ove				
	SIMD		vneghvh	Rb,Ra	W:ove				
ALU sum系	SIMD	ハーフワード	vsumh	Rb,Ra			Ra 15 0 → Rb	A	32
			vsumh2	Rb,Ra			Ra 15 0 15 0 → 15 0 Rb (+1) (+1) (丸め)		
		バイト	vabsaumb	Rc,Ra,Rb				C	
ALU その他	SIMD		fmdvh	Rb,Ra	W:ove		丸め	C	32
			vfmdvh	Rb,Rn	W:ove				
			vtel	Rc,Ra,Rb	R:VC			A	
			vsgnh	Rb,Ra				C	

【図２４】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
CMP	SINGLE		cmpCCn	Cm,Ra,Rb,Cn Cm,Ra,i05s,Cn Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05s,Cn	W:CF		CC = eq, ne, gt, ge, gtu, geu, lo, le, leu, ltu Cm <- result & Cm; (Cm+1 <- ~result & Cm;)	A	32
			cmpCCa	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05s,Cn	W:CF		Cm <- result & Cm; Cm+1 <- ~(result & Cm);		
			cmpCCo	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05s,Cn	W:CF		Cm <- result | Cm; Cm+1 <- ~(result | Cm);		
			cmpCC	C6,Ra2,Rb2 C6,Ra2,i04s	W:CF		CC = eq, ne, gt, ge, le, lt C6 <- result		16
			tstzn	Cm,Ra,Rb,Cn Cm,Ra,i05u,Cn Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb == 0) & Cm; (Cm+1 <- ~(Ra & Rb ==0) & Cm;)		32
			tstza	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb == 0) & Cm; Cm+1 <- ~((Ra & Rb == 0) & Cm);		
			tstzo	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb == 0) | Cm; Cm+1 <- ~((Ra & Rb == 0) | Cm);		
			tstnn	Cm,Ra,Rb,Cn Cm,Ra,i05u,Cn Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb != 0) & Cm; (Cm+1 <- ~(Ra & Rb != 0) & Cm;)		
			tstna	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb != 0) & Cm; Cm+1 <- ~((Ra & Rb != 0) & Cm);		
			tstno	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb != 0) | Cm; Cm+1 <- ~((Ra & Rb != 0) | Cm);		
			tstz	C6,Ra2,Rb2 C6,Ra2,i04u	W:CF		C6 <- (Ra2&Rb2 == 0)		16
			tstn	C6,Ra2,Rb2 C6,Ra2,i04u	W:CF		C6 <- (Ra2&Rb2 != 0)		
	SIMD	ハーフワード	vcmpCCh	Ra,Rb vcmpCCh Ra,i05s	W:CF		CC = eq, ne, gt, le, ge, lt		32
		バイト	vcmpCCb	Ra,Rb vacmpCCb Ra,i05s	W:CF		CC = eq, ne, gt, le ge, lt		

【図２５】


カテゴリ
SIMD
サイズ
命令
オペランド
CFR
PSR
代表的な動作
演算部
31
16
mul系
SINGLE
SIMD
ワード×ワード
ワード×ハーフワード
ハーフワード×ハーフワード
mul
mulu
mulhww
符号なし乗算
固定小数点演算
X2
X1
32
mulhw
mulhh
mulhhr
丸めあり
vmul
vfmulw
vfmulh
vfmulhr
vzmul
vxfmulw
vxfmulh
vxfmulhr
vfmul
vfxfmulw
vfxfmulh
vfxfmulhr
vcmul
vfcmulw
vfcmulh
vfcmulhr
vpfmulhww

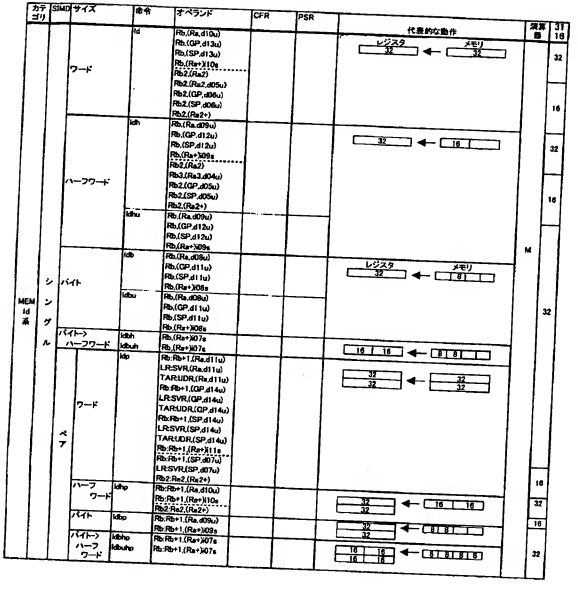
【図２６】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算静	31 16
msc系	SINGLE	ワード×ワード	mac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			mulを使った積和演算		
			fmscww	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	fmulwwを使った積和演算	X2	
		ワード×ハーフワード	hmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			hmulを使った積和演算		
			lmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			lmulを使った積和演算		
			fmachww	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	fmulhwwを使った積和演算	X1	
		ハーフワード×ハーフワード	fmachw	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	fmulhwを使った積和演算		
			fmachhr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	fmulhrを使った積和演算		
			fmachhv	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	丸めあり		32
	SIMD	ハーフワード×ハーフワード	vmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vmulを使った積和演算		
			vfmacw	Mm,Rc,Ra,Rb,Mn		fxp	vfmulwを使った積和演算		
			vxmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vxmulを使った積和演算		
			vcfmacw	Mm,Rc,Ra,Rb,Mn		fxp	vcfmulwを使った積和演算		
			vxfmach	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	vxfmulhを使った積和演算		
			vxfmachr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	丸めあり		
			vhmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vhmulを使った積和演算		
			vfhmacw	Mm,Rc,Ra,Rb,Mn		fxp	vfhmulwを使った積和演算		
			vfhmach	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	vfhmulhを使った積和演算	X2	
			vfhmachr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx					
			vlmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vlmulを使った積和演算		
			vflmacw	Mm,Rc,Ra,Rb,Mn		fxp	vflmulwを使った積和演算		
			vflmach	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	vflmulhを使った積和演算		
			vflmachr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	丸めあり		
			vflmach	Mm,Rc,Ra,Rb,Mn Mm,Rc,Ra,Rb,Mn		fxp	vflmulhを使った積和演算		
			vflmachr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	丸めあり		
		ワード×ハーフワード	vplmachww	Mm,Rc Rc+1,Ra,Rb,Mn		fxp			

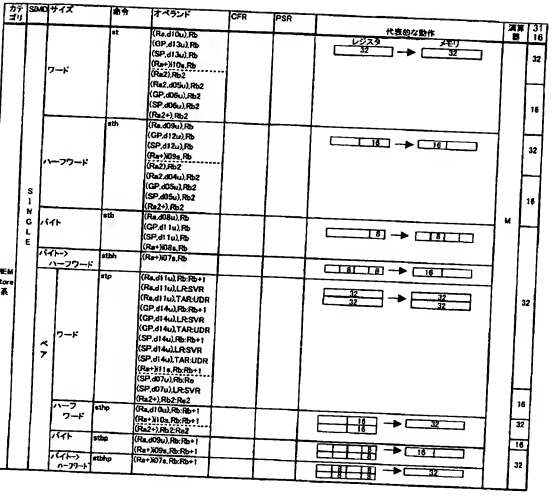
【図２７】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
msu系	SINGLE	ワード×ワード	msu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			msuを使った積差演算	X2	
			fmsuww	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	fmsuwwを使った積差演算		
		ワード×ハーフワード	lmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx Mm,Rc,Ra,Rb,Mh M0,Rc,Ra,Rb,Rx			lmsuを使った積差演算 lmulを使った積差演算	X1	
			fmsuhww	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	fmsuhwwを使った積差演算		
		ハーフワード×ハーフワード	fmsuhw	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	fmsuhwを使った積差演算		
			fmsuhh	Mm,Rc,Ra,Rb,Mh M0,Rc,Ra,Rb,Rx		fxp	fmsuhhを使った積差演算		
			fmsuhhr	Mm,Rc,Ra,Rb,Mh M0,Rc,Ra,Rb,Rx		fxp	丸めあり		
		ハーフワード×ハーフワード	vmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vmsuを使った積差演算	X2	32
			vfmsuw	Mm,Rc,Ra,Rb,Mn		fxp	vfmsuを使った積差演算		
			vfmsuh	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	vfmsuhを使った積差演算		
			vxmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vxmsuを使った積差演算		
			vxfmsuw	Mm,Rc,Ra,Rb,Mn		fxp	vxfmsuwを使った積差演算		
			vxfmsuh	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	vxfmsuhを使った積差演算		
			vhmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vhmsuを使った積差演算		
			vhfmsuw	Mm,Rc,Ra,Rb,Mn		fxp	vhfmsuwを使った積差演算		
			vhfmsuh	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	vhfmsuhを使った積差演算		
			vlmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vlmsuを使った積差演算		
			vlfmsuw	Mm,Rc,Ra,Rb,Mn		fxp	vlfmsuwを使った積差演算		
			vlfmsuh	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	vlfmsuhを使った積差演算		

【図２８】


カテゴリ
SIMD
サイズ
命令
オペランド
CFR
PSR
代表的な動作
演算器
31
16
MEM系
シングル
ペア
バイト
ワード
ハーフワード
バイト→ハーフワード
ld
ldh
ldhu
ldb
ldbu
ldbh
ldbuh
ldp
ldhp
ldbp
ldbhp
ldbuhp
Rb,(Ra,d10u)
Rb,(GP,d13u)
Rb,(SP,d13u)
Rb,(Ra+)10s
Rb2,(Ra2)
Rb2,(Ra2,d05u)
Rb2,(GP,d06u)
Rb2,(SP,d06u)
Rb2,(Ra2+)
Rb,(Ra,d09u)
Rb,(GP,d12u)
Rb,(SP,d12u)
Rb,(Ra+)09s
Rb,(Ra,d08u)
Rb,(GP,d11u)
Rb,(SP,d11u)
Rb,(Ra+)08s
Rb,(Ra,d08u)
Rb,(GP,d11u)
Rb,(SP,d11u)
Rb,(Ra+)08s
Rb,(Ra+)07s
Rb,(Ra+)07s
Rb:Rb+1,(Ra,d11u)
LR:SVR,(Ra,d11u)
TAR:UDR,(Ra,d11u)
Rb:Rb+1,(GP,d14u)
LR:SVR,(GP,d14u)
TAR:UDR,(GP,d14u)
Rb:Rb+1,(SP,d14u)
LR:SVR,(SP,d14u)
TAR:UDR,(SP,d14u)
Rb:Rb+1,(Ra+)11s
Rb:Rb+1,(SP,d07u)
LR:SVR,(SP,d07u)
Rb2:Ra2,(Ra2+)
Rb:Rb+1,(Ra,d10u)
Rb:Rb+1,(Ra+)10s
Rb2:Ra2,(Ra2+)
Rb:Rb+1,(Ra,d09u)
Rb:Rb+1,(Ra+)09s
Rb:Rb+1,(Ra+)07s
Rb:Rb+1,(Ra+)07s
レジスタ
メモリ
32
16
8
M

【図２９】


カテゴリ
SIMD
サイズ
命令
オペランド
CFR
PSR
代表的な動作
レジスタ
メモリ
演算器
31
16
SINGLE
MEM store 系
ワード
st
(Ra,d10u),Rb
(GP,d13u),Rb
(SP,d13u),Rb
(Re+)i10e,Rb
(Ra2),Rb2
(Ra2,d05u),Rb2
(GP,d06u),Rb2
(SP,d06u),Rb2
(Re2+),Rb2
ハーフワード
sth
(Ra,d09u),Rb
(GP,d12u),Rb
(SP,d12u),Rb
(Re+)i09e,Rb
(Ra2),Rb2
(Ra2,d04u),Rb2
(GP,d05u),Rb2
(SP,d05u),Rb2
(Re2+),Rb2
バイト
stb
(Ra,d08u),Rb
(GP,d11u),Rb
(SP,d11u),Rb
(Re+)i08e,Rb
M
バイト->ハーフワード
stbh
(Re+)i07e,Rb
ベア
ワード
stp
(Ra,d11u),Rb:Rb+1
(Ra,d11u),LR:SVR
(Ra,d11u),TAR:UDR
(GP,d14u),Rb:Rb+1
(GP,d14u),LR:SVR
(GP,d14u),TAR:UDR
(SP,d14u),Rb:Rb+1
(SP,d14u),LR:SVR
(SP,d14u),TAR:UDR
(Re+)i11e,Rb:Rb+1
(SP,d07u),Rb:Rb
(SP,d07u),LR:SVR
(Re2+),Rb2:Re2
ハーフワード
sthp
(Ra,d10u),Rb:Rb+1
(Re+)i10e,Rb:Rb+1
(Re2+),Rb2:Re2
バイト
sthp
(Ra,d09u),Rb:Rb+1
(Re+)i09e,Rb:Rb+1
バイト->ハーフワード
stbhp
(Re+)i07e,Rb:Rb+1

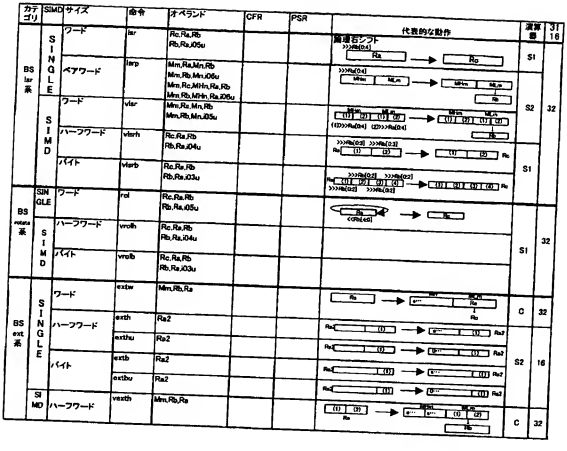
【図３０】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	3T 1B
BRA			setlr	d09s C5,d09s			LRを設定	8	32
			settar	d09s C6,d09s C6,C2:C4,d09s C6,Cm,d09s C6,C4,d09s	W:c6 W:c2:c4,c6 W:c6,cm W:c6		LRからフェッチした命令を分岐用バッファに格納 TARを設定 TARからフェッチした命令を分岐用バッファに格納		
			setlrb	LR			LRからフェッチした命令を分岐用バッファに格納		16
			setlb	TAR			TARからフェッチした命令を分岐用バッファに格納		
			loop	C5,LR,Ra,i08s C6,TAR,Ra,i08s C6,C2:C4,TAR,Ra,i08s C6,Cm,TAR,Ra,i08s C6,TAR,Rs2 C6,C2:C4,TAR,Rs2 C6,Cm,TAR,Rs2	W:c5 W:c6 W:c2:c4,c6 W:c6,cm W:c6 W:c2:c4,c6 W:c6		プレディケート[c5]のみ プレディケート[c6]のみ		32
			jmp	TAR LR					16
			jmpl	TAR LR	R:CF				
			jmpf	TAR LR Cm,TAR C6,C2:C4,TAR					
			jmpr	LR					
			br	d20s d09s			プレディケート[c6][c7]のみ		32
			brl	d20s d09s	R:CF				16 32
			rti			W:PSR R:ah			16

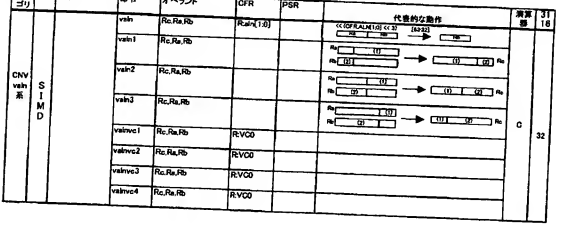
【図３１】

カテゴリ	SIMD	サイズ	令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
BS asl 系	SINGLE	ワード	asl	Rc,Ra,Rb Rb,Ra,i05u Ra2,i04u			左シフト <<Rb[0:4] Ra → Rc	S1	32
			faslvw	Rc,Ra,Rb Rb,Ra,i05u Rc,Ra,Rb Rb,Ra,i05u	W:ovs		<<Rb[0:4] Ra → 飽和あり → Rc		16
		ペアワード	aslp	Mm,Ra,Mn,Rb Mm,Rb,Mn,i06u Mm,Rc,MHn,Ra,Rb Mm,Rb,MHn,Ra,i06u			<<Ra[0:4] MHm MLm → MHm MLm Rb	S2	32
			faslpvw	Mm,Ra,Mn,Rb Mm,Rb,Mn,i06u	W:ovs		<<Rb[0:4] MHm MLm → 飽和あり → MHm MLm Rb		
	SIMD	ワード	vasl	Mm,Ra,Mn,Rb Mm,Rb,Mn,i05u			MHm (1) (2) MLm (1) (2) → MHm (1) (2) MLm (1) (2) (1)<<Ra[0:4] (2)<<Ra[0:4] Rb	S2	
			vfaslvw	Mm,Ra,Mn,Rb Mm,Rb,Mn,i05u	W:ovs		MHm (1) (2) MLm (1) (2) → 飽和あり → MHm (1) (2) MLm (1) (2) (1)<<Ra[0:4] (2)<<Ra[0:4] Rb		
		ハーフワード	vaslh	Rc,Ra,Rb Rb,Ra,i04u			<<Rb[0:3] <<Rb[0:3] Ra (1) (2) → (1) (2) Rc	S1	
			vfaslvh	Rc,Ra,Rb Rb,Ra,i04u	W:ovs		<<Rb[0:3] <<Rb[0:3] Ra (1) (2) → 飽和あり → (1) (2) Rc		
		バイト	vaslb	Rc,Ra,Rb Rb,Ra,i03u			<<Rb[0:2] <<Rb[0:2] Ra (1) (2) (3) (4) → (1) (2) (3) (4) Rc <<Rb[0:2] <<Rb[0:2]		
BS asr 系	SINGLE	ワード	asr	Rc,Ra,Rb Rb,Ra,i05u Ra2,i04u			算術右シフト >>Rb[0:4] Ra → Rc	S1	32
		ペアワード	asrp	Mm,Ra,Mn,Rb Mm,Rb,Mn,i06u Mm,Rc,MHn,Ra,Rb Mm,Rb,MHn,Ra,i06u			>>Ra[0:4] MHm MLm → MHm MLm Rb	S2	16
	SIMD	ワード	vasr	Mm,Ra,Mn,Rb Mm,Rb,Mn,i05u			MHm (1) (2) MLm (1) (2) → MHm (1) (2) MLm (1) (2) (1)>>Ra[0:4] (2)>>Ra[0:4] Rb		32
		ハーフワード	vasrh	Rc,Ra,Rb Rb,Ra,i04u			>>Rb[0:3] >>Rb[0:3] Ra (1) (2) → (1) (2) Rc	S1	
		バイト	vasrb	Rc,Ra,Rb Rb,Ra,i03u			>>Rb[0:2] >>Rb[0:2] Ra (1) (2) (3) (4) → (1) (2) (3) (4) Rc >>Rb[0:2] >>Rb[0:2]		

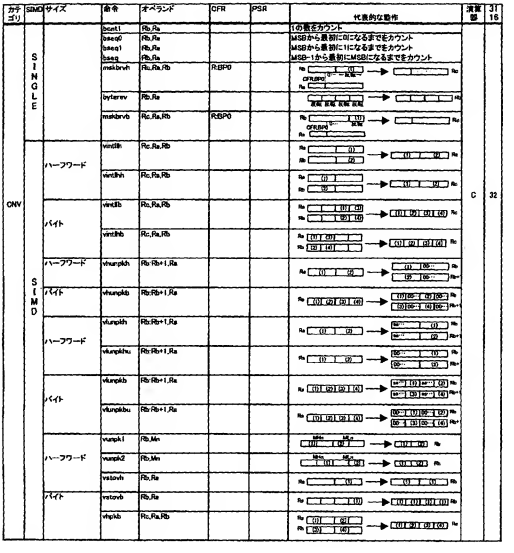
【図３２】


カテゴリ
SIMD
サイズ
命令
オペランド
CFR
PSR
代表的な動作
演算器
31
16
BS
bir
系
SINGLE
SIMD
ワード
bir
Rc,Ra,Rb
Rb,Ra,i05u
ビアワード
birp
Mm,Ra,Mn,Rb
Mm,Rb,Mn,i05u
Mm,Rc,Mfrh,Ra,Rb
Mm,Rb,Mfrh,Ra,i05u
ワード
vbir
Mm,Rc,Mn,Rb
Mm,Rb,Mn,i05u
ハーフワード
vbirh
Rc,Ra,Rb
Rb,Ra,i04u
バイト
vbirb
Rc,Ra,Rb
Rb,Ra,i03u
論理右シフト
S1
S2
32
BS
rotate
系
SINGLE
SIMD
ワード
rol
Rc,Ra,Rb
Rb,Ra,i05u
ハーフワード
vrolh
Rc,Ra,Rb
Rb,Ra,i04u
バイト
vrolb
Rc,Ra,Rb
Rb,Ra,i03u
S1
32
BS
ext
系
SINGLE
ワード
extw
Mm,Rb,Ra
ハーフワード
exth
Ra2
exthu
Ra2
バイト
extb
Ra2
extbu
Ra2
SIMD
ハーフワード
vexth
Mm,Rb,Ra
C
S2
32
16

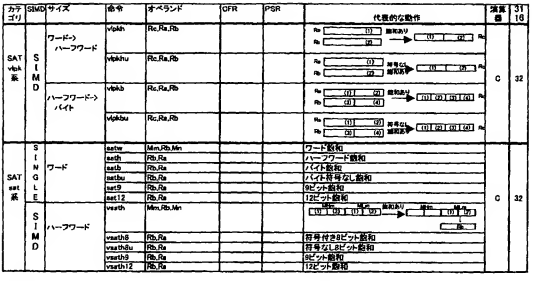
【図３３】


カテゴリ
SIMD
サイズ
命令
オペランド
CFR
PSR
代表的な動作
演算器
31
16
CNV
vsin
系
SIMD
vsin
Rc,Ra,Rb
R:sin(1.0)
vsin1
Rc,Ra,Rb
vsin2
Rc,Ra,Rb
vsin3
Rc,Ra,Rb
vsinvc1
Rc,Ra,Rb
R:VC0
vsinvc2
Rc,Ra,Rb
R:VC0
vsinvc3
Rc,Ra,Rb
R:VC0
vsinvc4
Rc,Ra,Rb
R:VC0
C
32

【図３４】

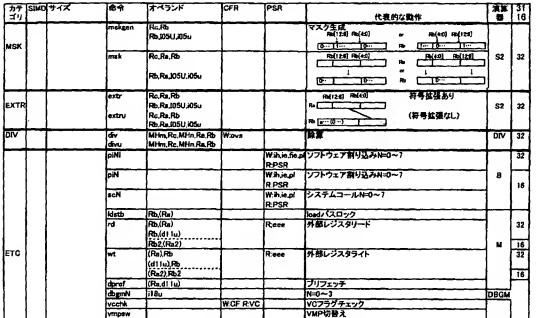

カテゴリ
SIMD
サイズ
命令
オペランド
CFR
PSR
代表的な動作
演算器
31 16
CNV
SINGLE
SIMD
bcnt1 Rb,Ra
bseq0 Rb,Ra
bseq1 Rb,Ra
bseq Rb,Ra
1の個数をカウント
MSBから最初に0になるまでをカウント
MSBから最初に1になるまでをカウント
MSB-1から最初にMSBになるまでをカウント
maskbrvh Rc,Ra,Rb R:BP0
byterev Rb,Ra
maskbrvb Rc,Ra,Rb R:BP0
vintlh Rc,Ra,Rb
ハーフワード
vintlhh Rc,Ra,Rb
vintlb Rc,Ra,Rb
バイト
vintlhb Rc,Ra,Rb
vhunpkh Rb,Rb+1,Ra
ハーフワード
vhunpkb Rb,Rb+1,Ra
バイト
vfunpkh Rb,Rb+1,Ra
ハーフワード
vfunpkhu Rb,Rb+1,Ra
vfunpkb Rb,Rb+1,Ra
バイト
vfunpkbu Rb,Rb+1,Ra
vumpk1 Rb,Mn
ハーフワード
vumpk2 Rb,Mn
vsovh Rb,Ra
バイト
vsovb Rb,Ra
vhpkb Rc,Ra,Rb
C
32

【図３５】



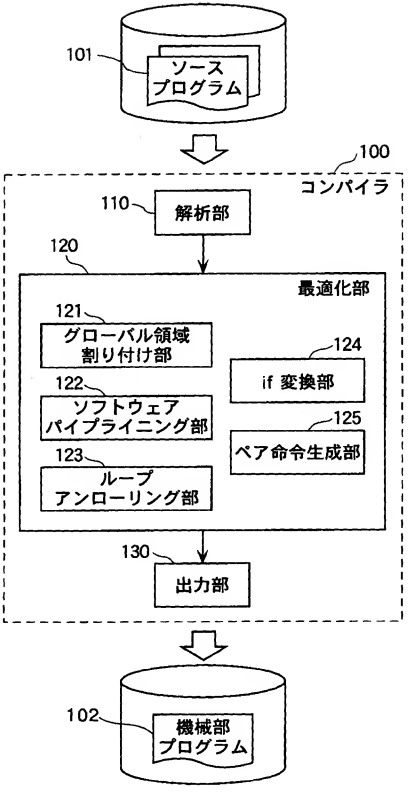
カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
SAT vsk 系	SIMD	ワード→ハーフワード	vpkh	Rc,Ra,Rb				C	32
			vpkhu	Rc,Ra,Rb					
		ハーフワード→バイト	vpkb	Rc,Ra,Rb					
			vpkbu	Rc,Ra,Rb					
SAT sat 系	SINGLE	ワード	satw	Mm,Rb,Mn			ワード飽和	C	32
			sath	Rb,Ra			ハーフワード飽和		
			satb	Rb,Ra			バイト飽和		
			satbu	Rb,Ra			バイト符号なし飽和		
			sat9	Rb,Ra			9ビット飽和		
			sat12	Rb,Ra			12ビット飽和		
	SIMD	ハーフワード	vsath	Mm,Rb,Mn					
			vsath8	Rb,Ra			符号付き8ビット飽和		
			vsathBu	Rb,Ra			符号なし8ビット飽和		
			vsath9	Rb,Ra			9ビット飽和		
			vsath12	Rb,Ra			12ビット飽和		

【図３６】

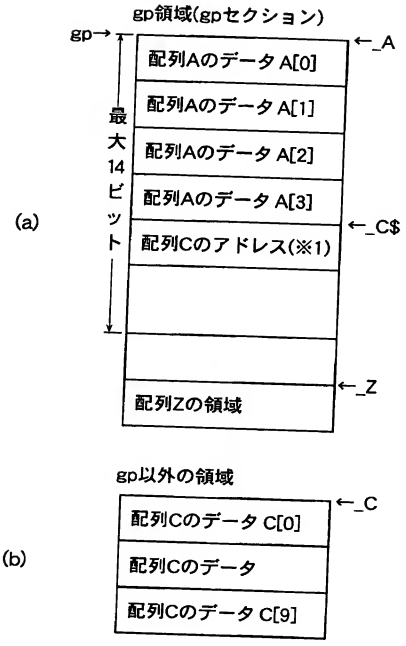


カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
MSK			mskgen	Rc,Rb Rb,I05U,I05u			マスク生成 Rb[12:8] Rb[4:0] or Rb[4:0] Rb[12:8]		
			msk	Rc,Ra,Rb			Rb[12:8] Rb[4:0] Rb[4:0] Rb[12:8]	S2	32
				Rb,Ra,I05U,I05u					
EXTR			extr	Rc,Ra,Rb Rb,Ra,I05U,I05u			Rb[12:8] Rb[4:0] 符号拡張あり	S2	32
			extru	Rc,Ra,Rb Rb,Ra,I05U,I05u			(符号拡張なし)		
DIV			div divu	Mm,Rc,Mn,Ra,Rb Mm,Rc,Mn,Ra,Rb	Wdiva		除算	DIV	32
ETC			pdN			W:sh,le,se,pl R:PSR	ソフトウェア割り込みN=0～7	B	32 16
			piN			W:sh,ie,pl R:PSR	ソフトウェア割り込みN=0～7		
			scN			W:sh,ie,pl R:PSR	システムコールN=0～7		
			ldstb	Rb,(Ra)			load/ストロック	M	32 16 32 16
			rd	Rb,(Ra) Rb,(d11u), Rb2,(Ra2)		R:eee	外部レジスタリード		
			wt	(Ra),Rb (d11u),Rb, (Ra2),Rb2		R:eee	外部レジスタライト		
			dpref	(Ra,d11u)			プリフェッチ		
			dbgnN	I18u			N=0～3	DBGM	
			vcchk		WCF,RVC		VCフラグチェック	B	32
			vmpsw				VMP切替え		
			vmpsw	LR					
			vmpimd1			W:ie	VMP切替え禁止		
			vmpintd2						
			vmpintd3			W:ie			
			vmpinte1			W:ie	VMP切替え許可		
			vmpinte2						
			vmpinte3						
			nop				no operation	A	16

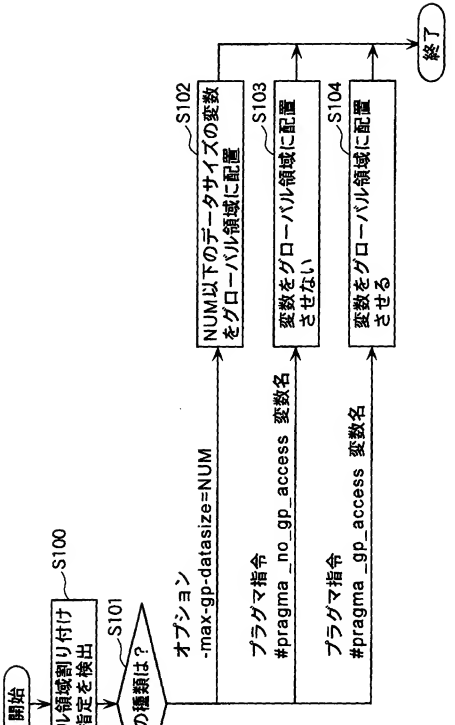
【図３７】


101
ソース
プログラム
100
コンパイラ
110
解析部
120
最適化部
121
グローバル領域
割り付け部
122
ソフトウェア
パイプライニング部
123
ループ
アンローリング部
124
if 変換部
125
ペア命令生成部
130
出力部
102
機械部
プログラム

【図３８】


gp領域(gpセクション)
gp→
最大14ビット
配列Aのデータ A[0]
配列Aのデータ A[1]
配列Aのデータ A[2]
配列Aのデータ A[3]
配列Cのアドレス(※1)
配列Zの領域
←_A
←_C$
←_Z
(a)
gp以外の領域
配列Cのデータ C[0]
配列Cのデータ
配列Cのデータ C[9]
←_C
(b)

【図３９】


開始
グローバル領域割り付けに関する指定を検出
S100
S101
指定の種類は？
オプション
-max-gp-datasize=NUM
プラグマ指令
#pragma _no_gp_access 変数名
プラグマ指令
#pragma _gp_access 変数名
S102
NUM以下のデータサイズの変数をグローバル領域に配置
S103
変数をグローバル領域に配置させない
S104
変数をグローバル領域に配置させる
終了

【図４０】

最大データサイズの変化による生成コードの比較

デフォルト	-mmax-gp-datasize=40を指定

inta[8];
intc[10];
 	
intsample(void)
{
a[7]=c[0]+c[9];
 	
returna[7];
}
	inta[8];
	intc[10];
	
	intsample(void)
	{
	a[7]=c[0]+c[9];
	
	returna[7];
	}

ldr1,(gp,_c$-.MN.gptop);;
setlor0,LO(_c+36);;
sethir0,HI(_c+36);;
ld r1,(r1);;
ld r0,(r0);;
addr0,r1,r0;;
st(gp,_a-.MN.gptop+28),r0
ret;;
	ld r1,(gp,_c-.MN.gptop);;
	ldr0,(gp,_c-.MN.gptop+36);;
	
	add r0,r1,r0;;
	st (gp,_a-.MN.gptop+28),r0
	ret;;

10サイクル 8バイト	7サイクル 5バイト

【図４１】

gp領域配置の最大データサイズを40とした場合

サイズ指定なしファイル外定義	ファイル内定義／ サイズ指定ファイル外定義

extern int a[ ];
extern int c[ ];
 	
int sample(void)
{
a[7] = c[0] + c[9];
 	
return a[7];
}
	int a[8];
	extern int c[10];
	
	int sample(void)
	{
	a[7] = c[0] + c[9];
	
	return a[7];
	}

setlo r0,LO(_c);;
setlo r1,LO(_c+36)
sethi r0,HI(_c);;
sethi r1,HI(_c+36);;
ld r3,(r0);;
setlo r2,LO(_a+28)
ld r0,(r1);;
sethi r2,HI(_a+28);;
add r1,r3,r0;;
mov r0,r1
st (r2),r1
ret ;;
	ld r1,(gp,_c - .MN.gptop);;
	
	ld r0,(gp,_c - .MN.gptop+36);;
	
	add r0,r1,r0;;
	
	st (gp,_a - .MN.gptop+28),r0
	ret;;

10サイクル 12バイト	7サイクル 5バイト

【図４２】

gp領域配置の最大データサイズをデフォルトの32とした場合

サイズ指定ありファイル外定義／ファイル内定義 #pragma _no_gp_access指令	サイズ指定なしファイル外定義／ファイル内定義 #pragma _gp_access指令

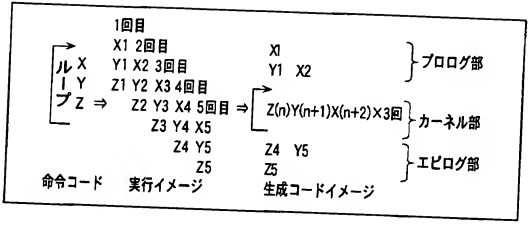
#pragma _no_gp_access a, c
 	
extern int a[8];
int c[10];
 	
int sample(void)
{
a[7] = c[0] + c[9];
 	
return a[7];
}
	#pragma _gp_access a, c
	
	extern int a[];
	int c[10];
	
	int sample(void)
	{
	a[7] = c[0] + c[9];
	
	return a[7];
	}

setlo r0,LO(_c);;
setlo r1,LO(_c+36)
sethi r0,HI(_c);;
sethi r1,HI(_c+36);;
ld r3,(r0);;
setlo r2,LO(_a+28)
ld r0,(r1);;
sethi r2,HI(_a+28);;
add r1,r3,r0;;
mov r0,r1
st (r2),r1
ret ;;
	ld r1,(gp,_c - .MN.gptop);;
	
	ld r0,(gp,_c - .MN.gptop+36);;
	
	add r0,r1,r0;;
	
	st (gp,_a - .MN.gptop+28),r0
	ret;;

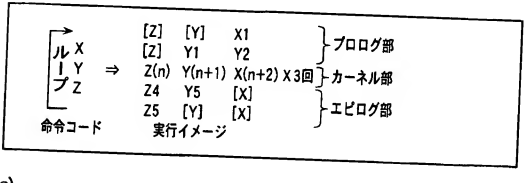
10サイクル 12バイト	7サイクル 5バイト

【図４３】

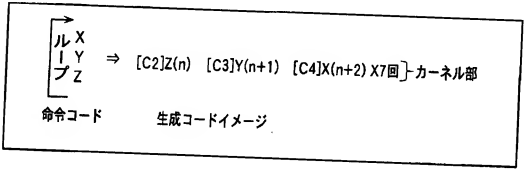
(a)


1回目
X1 2回目
Y1 X2 3回目
Z1 Y2 X3 4回目
Z2 Y3 X4 5回目
Z3 Y4 X5
Z4 Y5
Z5
ループX Y Z
⇒
X1
Y1 X2
Z(n)Y(n+1)X(n+2)×3回
Z4 Y5
Z5
プロログ部
カーネル部
エピログ部
命令コード
実行イメージ
生成コードイメージ

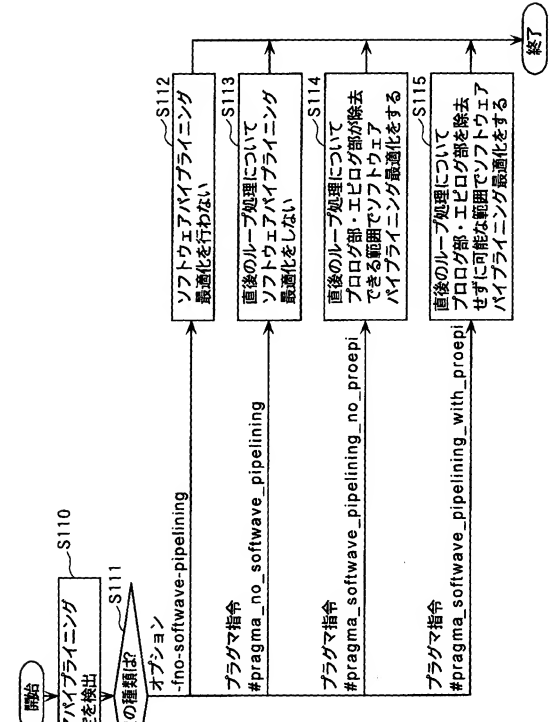
(b)


ループX Y Z
⇒
[Z] [Y] X1
[Z] Y1 Y2
Z(n) Y(n+1) X(n+2) X3回
Z4 Y5 [X]
Z5 [Y] [X]
プロログ部
カーネル部
エピログ部
命令コード
実行イメージ

(c)


ループX Y Z
⇒
[C2]Z(n) [C3]Y(n+1) [C4]X(n+2) X7回
カーネル部
命令コード
生成コードイメージ

【図４４】


開始
ソフトウェアパイプライニング
に関する指定を検出
S110
指定の種類は?
S111
オプション
-fno-softwave-pipelining
プラグマ指令
#pragma_no_softwave_pipelining
プラグマ指令
#pragma_softwave_pipelining_no_proepi
プラグマ指令
#pragma_softwave_pipelining_with_proepi
ソフトウェアパイプライニング
最適化を行わない
S112
直後のループ処理について
ソフトウェアパイプライニング
最適化をしない
S113
直後のループ処理について
プロログ部・エピログ部が除去
できる範囲でソフトウェア
パイプライニング最適化をする
S114
直後のループ処理について
プロログ部・エピログ部を除去
せずに可能な範囲でソフトウェア
パイプライニング最適化をする
S115
終了

【図４５】

コンパイルオプション-Oでのコンパイル

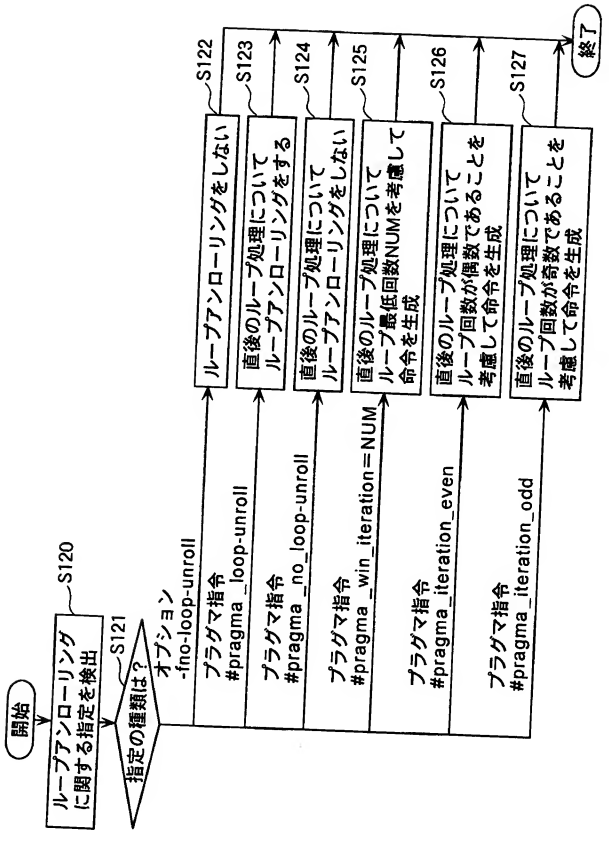
デフォルト	#pragma _software_pipelining_with_proepi 指令

int a[100];
int b[100];
void sample(int c)
{
int i;
 	
for(i=0; i<100; i++) {
a[i] = b[i] + c;
}
}
	int a[100];
	int b[100];
	void sample(int c)
	{
	int i;
	#pragma _software_pipelining_with_proepi
	for(i=0; i<100; i++) {
	a[i] = b[i] + c;
	}
	}

ld r5,(gp,_x$ - .MN.gptop);;
 	
mov r1,98
settar C6,C4,L00030
ld r4,(gp,_v$ - .MN.gptop);;
L00030
[C4] add r2,r3,r0
[C6] ld r3,(r5+);;
[C4] st (r4+),r2
[C6] jloop C6,C4,tar,r1,-1;;
 	
ret ;;
	ld r5,(gp,_x$ - .MN.gptop);;
	ld r4,(r5+);;
	mov r1,96
	settar C6,L00023
	ld r3,(gp,_v$ - .MN.gptop);;
	L00023
	add r2,r4,r0
	ld r4,(r5+);;
	st (r3+),r2
	[C6] jloop C6,C4,tar,r1,-1;;
	add r2,r4,r0;;
	st (r3+),r2
	ret ;;

2+2×101+3=207サイクル 9バイト	3+2×99+3=204サイクル 12バイト

【図４６】


開始
ループアンローリング
に関する指定を検出
S120
指定の種類は？
S121
オプション
-fno-loop-unroll
プラグマ指令
#pragma _loop-unroll
プラグマ指令
#pragma _no_loop-unroll
プラグマ指令
#pragma _win_iteration＝NUM
プラグマ指令
#pragma_iteration_even
プラグマ指令
#pragma_iteration_odd
ループアンローリングをしない
S122
直後のループ処理について
ループアンローリングをする
S123
直後のループ処理について
ループアンローリングをしない
S124
直後のループ処理について
ループ最低回数NUMを考慮して
命令を生成
S125
直後のループ処理について
ループ回数が偶数であることを
考慮して命令を生成
S126
直後のループ処理について
ループ回数が奇数であることを
考慮して命令を生成
S127
終了

【図４７】

#pragma _loop_unroll指令の実行速度向上例

デフォルト	#pragma _loop_unroll指令

int a[100];
int b[100];
void sample(int c)
{
int i, ret=0;
 	
for(i=0; i<100; i++) {
ret += a[i] * a[i];
}
return ret;
}
	int a[100];
	int b[100];
	void sample(int c)
	{
	int i, ret=0;
	#pragma _loop_unroll
	for(i=0; i<100; i++) {
	ret += a[i] * a[i];
	}
	return ret;
	}

mov r0,0
mov r1,98;;
settar C6,C4,L00013
 	
ld r3,(gp,_a$ - .MN.gptop)
mul m0,r2,gp,0;;
L00013
[C4] mac m0,r0,r2,r2,m0
 	
[C6] ld r2,(r3+)
 	
[C6] jloop C6,C4,tar,r1,-1;;
ret ;;
	ld r3,(gp,_a$ - .MN.gptop);;
	mov r0,0
	mov r1,48;;
	settar C6,C2:C4,L00013
	add r5,r3,4
	
	mul m0,r2,gp,0;;
	L00013
	[C2] mac m0,r0,r2,r2,m0
	[C3] add r3,r3,8
	[C3] ld r2,(r5+)8;;
	[C3] mac m0,r0,r4,r4,m0
	[C4] ld r4,(r3)
	[C6] jloop C6,C2:C4,tar,r1,-1;;
	ret ;;

2＋2×101＋3=207サイクル 9バイト	3＋2×52＋3＝110サイクル 13バイト

【図４８】

デフォルト	#pragma _loop_unroll指令 と #pragma _align_local_pointer指令

int a[100];
int b[100];
void sample(int c)
{
　int i;
 	
　int ＊pa=a;
　int ＊pb=b;
 	
　for(i=0; i<100; i++) {
　　＊pa++ = i ＊ c >> ＊pb++;
　}
}
	int a[100];
	int b[100];
	void sample(int c)
	{
	　int i;
	#pragma _align_local_pointer=8 pa,pb
	　int ＊pa=a;
	　int ＊pb=b;
	#pragma _loop_unroll
	　for(i=0; i<100; i++) {
	　　＊pa++ = i ＊ c >> ＊pb++;
	　}
	}

mov r4,0
　ld r6,(gp,_a$ - .MN.gptop);;
　mov r1,98
　settar C6,C4,L00025
　ld r5,(gp,_b$ - .MN.gptop);;
L00025
[C4] asr r2,r4,r3;;
 	
[C6] ld r3,(r5+);;
[C4] add r4,r4,r0
[C4] st (r6+),r2
[C6] jloop C6,C4,tar,r1,-1;;
　ret ;;
	mov r6,0
	　ld r8,(gp,_a$ - .MN.gptop);;
	　mov r1,48
	　settar C6,C4,L00016
	　ld r7,(gp,_b$ - .MN.gptop);;
	L00016
	[C2] add r6,r6,r0;;
	[C2] stp (r8+),r2:r3
	[C3] asr r2,r6,r4
	[C3] add r6,r6,r0;;
	[C3] asr r3,r6,r5
	[C4] ldp r4:r5,(r7+)
	[C6] jloop C6,C2:C4,tar,r1,-1;;
	　ret ;;

2＋3×101＋3=308サイクル 11バイト	2＋3×51＋3＝158サイクル 13バイト

【図４９】

#pragma _min_iteration指令なし	#pragma _min_iteration指令あり

int a[101];
int b[101];
void sample(int c, int end)
{
int i;
int ＊pa=a;
int ＊pb=b;
 	
for (i = 0; i <end; i++) {
＊pa++ = i ＊ c >> ＊pb++;
}
＊pa = end;
}
	int a[101];
	int b[101];
	void sample(int c, int end)
	{
	int i;
	int ＊pa=a;
	int ＊pb=b;
	#pragma _min_iteration=4
	for (i = 0; i <end; i++) {
	＊pa++ = i ＊ c >> ＊pb++;
	}
	＊pa = end;
	}

cmple C0,r1,0
ld r5,(gp,_a$ - .MN.gptop);;
mov r4,0
ld r6,(gp,_b$ - .MN.gptop)
[C0] br L00016;;
mov r3,0
settar L00017
L00017
ld r2,(r6+);;
add r4,1;;
cmplt C0,r4,r1
asr r2,r3,r2;;
add r3,r3,r0
st (r5+),r2
[C0] jmpf tar;;
L00016
st (r5),r1
ret ;;
	mov r5,0
	ld r7,(gp,_a$ - .MN.gptop);;
	settar C6,C4,L00027
	sub r2,r1,2
	ld r6,(gp,_b$ - .MN.gptop);;
	
	L00027
	[C4] asr r3,r5,r4;;
	[C6] ld r4,(r6+);;
	[C4] add r5,r5,r0
	[C4] st (r7+),r3
	[C6] jloop C6,C4,tar,r2,-1;;
	
	st (r7),r1
	ret ;;

2＋4×100＋3=405サイクル 16バイト	2＋3×101＋3=308サイクル 12バイト

【図５０】

ループアンローリング最適化が不可能な例

ループ回数が不明なC言語ソース	コンパイル結果

void sample(int c, int end)
{
　int i;
#pragma _align_local_pointer=8 pa, pb
　int *pa=a;
　int *pb=b;
#pragma _loop_unroll
#pragma _min_iteration=4
　for (i = 0; i <end; i++) {
　　*pa++ = i *c >> *pb++;
　}
　*pa = end;
}
	mov r5,0
	　　ld r7,(gp,_a$ - .MN.gptop);;
	　　settar C6,C4,L00026
	　　sub r2,r1,2
	　　ld r6,(gp,_b$ - .MN.gptop);;
	L00026
	[C4] asr r3,r5,r4;;
	[C6] ld r4,(r6+);;
	[C4] add r5,r5,r0
	[C4] st (r7+),r3
	[C6] jloop C6,C4,tar,r2,-1;;
	　　st (r7),r1
	　　ret ;;

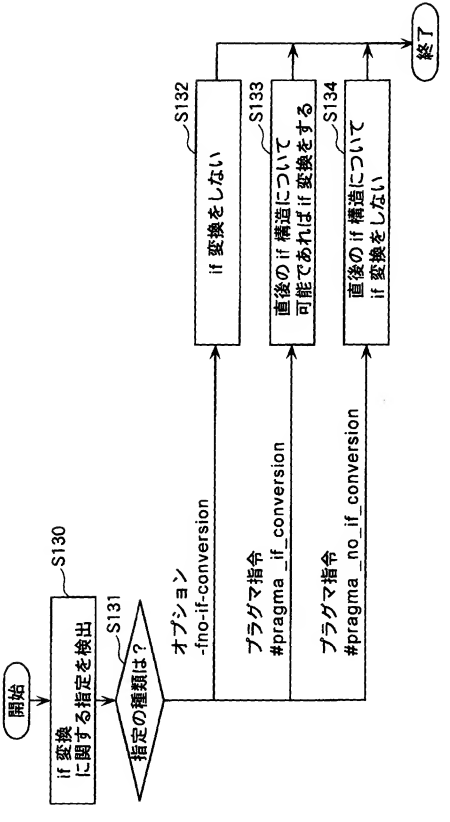
【図５１】

#pragma _iteration_even指令	#pragma _iteration_odd指令

int a[101];
int b[101];
void sample(int c, int end)
{
int i;
#pragma _align_local_pointer=8 pa, pb
int *pa=a;
int *pb=b;
#pragma _min_iteration=50
#pragma _loop_unroll
#pragma _iteration_even
for (i = 0; i <end; i++) {
*pa++ = i * c >> *pb++;
}
*pa = end;
}
	int a[101];
	int b[101];
	void sample(int c, int end)
	{
	int i;
	#pragma _align_local_pointer=8 pa, pb
	int *pa=a;
	int *pb=b;
	#pragma _min_iteration=50
	#pragma _loop_unroll
	#pragma _iteration_odd
	for (i = 0; i <end; i++) {
	*pa++ = i * c >> *pb++;
	}
	*pa = end;
	}

sub r2,r1,6;;
mov r6,0
asr r2,1
ld r7,(gp,_a$ - .MN.gptop);;
settar C6,C4,L00036
add r2,1
ld r8,(gp,_b$ - .MN.gptop);;
L00036
[C4] asr r3,r6,r4;;
[C4] add r6,r6,r0
[C4] st (r7+),r3;;
[C4] asr r3,r6,r5;;
[C6] ldp r4:r5,(r8+);;
[C4] add r6,r6,r0
[C4] st (r7+),r3
[C6] jloop C6,C4,tar,r2,-1;;
 	
st (r7),r1
ret ;;
	sub r2,r1,7;;
	mov r6,0
	asr r2,1
	ld r7,(gp,_a$ - .MN.gptop);;
	settar C6,C4,L00046
	add r2,1
	ld r8,(gp,_b$ - .MN.gptop);;
	L00046
	[C4] asr r3,r6,r4;;
	[C4] add r6,r6,r0
	[C4] st (r7+),r3;;
	[C4] asr r3,r6,r5;;
	[C6] ldp r4:r5,(r8+);;
	[C4] add r6,r6,r0
	[C4] st (r7+),r3
	[C6] jloop C6,C4,tar,r2,-1;;
	ld r2,(r8+);;
	asr r2,r6,r2;;
	st (r7+),r2;;
	st (r7),r1
	ret ;;

【図５２】


開始
if 変換
に関する指定を検出
S130
S131
指定の種類は？
オプション
-fno-if-conversion
プラグマ指令
#pragma _if_conversion
プラグマ指令
#pragma _no_if_conversion
S132
if 変換をしない
S133
直後の if 構造について
可能であれば if 変換をする
S134
直後の if 構造について
if 変換をしない
終了

【図５３】

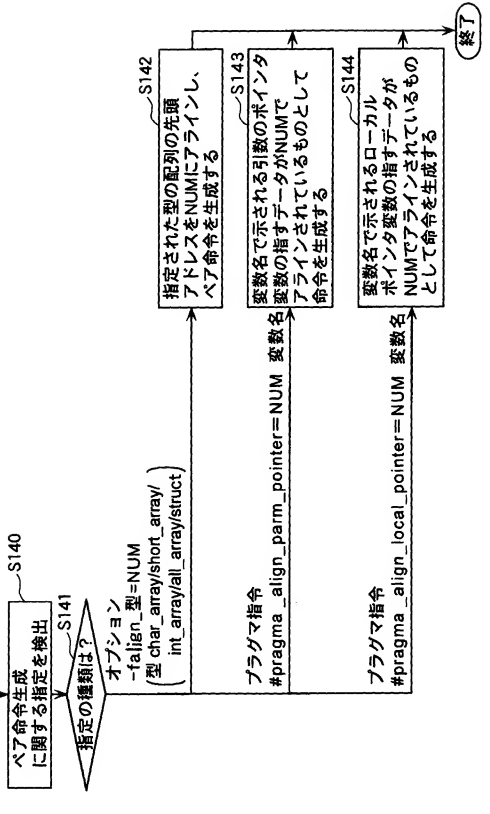
#pragma_no_if_conversion指令	#pragma_no_if_conversion指令

int func(int a)
{
int b ;
#pragma_no_if_conversion
if(a == 2)
b = 0 ;
else
b = 1 ;
return b ;
}
	int func(int a)
	{
	int b ;
	#pragma_if_conversion
	if(a == 2)
	b = 0 ;
	else
	b = 1 ;
	return b ;
	}

cmpne C0,r0,2 ;;
[C0] br L00002 ;; //分岐が生成されます
mov r0,0
ret ;;
L00002
mov r0,1
ret ;; ;;
	cmpeq C0:C1,r0,2 ;;
	[C0] mov r0,0 //分岐命令の変わりに、
	[C1] mov r0,1 //条件付き実行命令が生成されます
	ret ;;

then側5サイクル, else側7サイクル 12バイト;	4サイクル 8バイト

【図５４】


開始
ペア命令生成
に関する指定を検出
S140
S141
指定の種類は？
オプション
-falign_型=NUM
(型 char_array/short_array/
int_array/all_array/struct)
S142
指定された型の配列の先頭
アドレスをNUMにアラインし、
ペア命令を生成する
S143
変数名で示される引数のポインタ
変数の指すデータがNUMで
アラインされているものとして
命令を生成する
S144
変数名で示されるローカル
ポインタ変数の指すデータが
NUMでアラインされているもの
として命令を生成する
終了
プラグマ指令
#pragma _align_parm_pointer=NUM 変数名
プラグマ指令
#pragma _align_local_pointer=NUM 変数名

【図５５】

「アライン指定なし」	「オプション-falign-short-array=4指定」

short src[8] ;
short dst[8] ;
 	
void align1(void)
{
　　char　dummy ;
　　int　　i ;
 	
　　for(i=0 ; i<8 ; i+=2)
　　　　dst[i] = src[i] + src[i+1] ;
}
	変更なし

addu r5,gp,_src - .MN.gptop
　　addu r4,gp,_src - .MN.gptop+2 ;;
　　addu r3,gp,_dst - .MN.gptop
　　mov　r0,2
　　settar C6,C4,L00005 ;;
L00005
[C4]　ldh　　r1,(r4+)4 ;; //アラインメント不明のため、
[C4]　add　r1,r2,r1　　//２つデータは独立して
[C6]　ldh　　r2,(r5+)4 ;; //ロードされます(ldh命令)
[C4]　sth　　(r3+)4,r1
[C6]　jloop　C6,C4,tar,r0,-1 ;; //L00005
　　ret　　;;
	addu r5,gp,_src - .MN.gptop ;;
	　　addu r4,gp,_dst - .MN.gptop
	　　mov　r0,2
	　　settar C6,C4,L00005 ;;
	L00005
	[C4]　add　r1,r2,r3　　//アラインメント4のため、
	[C6]　ldhp　r2:r3,(r5+) ;;　　//ペアアクセス命令(ldhp)
	[C4]　sth　　(r4+)4,r1　　//が生成されます。
	[C6]　jloop　C6,C4,tar,r0,-1 ;; //L00005
	　　ret　　;;

2+4X5+3=　25サイクル 22バイト	2+2X5+3=15サイクル 18バイト

【図５６】

「アライン指定なし」	「#pragma _align_parm_pointer 指令使用」

int align2(short * src)
{
int v = 0 ;
for(int i=0 ; i<100 ; i+=2){
v += src[i] * src[i+1] ;
}
return v ;
}
	#pragma _align_parm_pointer=4 src
	int align2(short * src)
	{
	int v = 0 ;
	for(int i=0 ; i<100 ; i+=2){
	v += src[i] * src[i+1] ;
	}
	return v ;
	}

mov r5,0
mov r1,48
settarC6,C4,L00005 ;;
mov r4,r0
add r3,r0,2
mul m0,r2,gp,0 ;;
L00005
[C4] ldh r0,(r3+)4 ;; //ソフトウェアパイプライニング適用
[C4] lmac m0,r5,r2,r0,m0 //データ07ライメントで制約とも
[C6] ldh r2,(r4+)4 // ldh命令を2回実行
[C6] jloop C6,C4,tar,r1,-1 ;; //L00005
mov r0,r5
ret ;;
	mov r4,0 ;;
	mov r1,48
	settarC6,C4,L00005
	mul m0,r2,gp,0 ;;
	L00005
	[C4] lmac m0,r4,r2,r3,m0 //ソフトウェアパイプライニング適用
	[C6] ldhp r2:r3,(r0+) //ペアアクセス命令使用可能。
	[C6] jloop C6,C4,tar,r1,-1 ;; //L00005
	mov r0,r4
	ret ;;

2+3×51+3=160サイクル 24バイト	2+2×51+3=107サイクル 18バイト

【図５７】

「アライン指令なし」	「#pragma _align_local_pointer指令使用」

void align3(int n)
{
short * from ;
short * to ;
int i ;
 	
from = &(src[n]) ; to = &(dst[n]) ;
for(i=0 ; i<16 ; i++, from+=2, to+=2){
* to = * from ;
* (to+1) = * (from+1) ;
}
}
	void align3(int n)
	{
	# pragma _align_local_pointer=4 from,to
	short * from ;
	short * to ;
	int i ;
	
	from = &(src[n]) ; to = &(dst[n]) ;
	for(i=0 ; i<16 ; i++, from+=2, to+=2){
	* to = * from ;
	* (to+1) = * (from+1) ;
	}
	}

ld r3,(gp,_src$ - .MN.gptop) ;;
ld r2,(gp,_dst$ - .MN.gptop) ;;
add r1,r0,r0 ;;
mov r0,13
add r6,r3,r1
add r5,r2,r1 ;;
settar C6,L00016
add r4,r5,2
add r3,r6,2 ;;
L00016
ldh r1,(r6+)4 ;;
ldh r2,(r3+)4 ;;
sth (r5+)4,r1 ;;
sth (r4+)4,r2
[C6] jloop C6,tar,r0,-1 ;; //L00016
ret ;;
	ld r3,(gp,_src$ - .MN.gptop) ;;
	ld r2,(gp,_dst$ - .MN.gptop) ;;
	mov r1,13
	add r0,r0,r0 ;;
	settar C6,L00016
	add r4,r3,r0
	add r0,r2,r0 ;;
	L00016
	ldhp r2:r3,(r4+) ;;
	sthp (r0+),r2:r3
	[C6] jloop C6,tar,r1,-1 ;; //L00016
	ret ;;

5+4×16+3=72サイクル 30バイト	5+3×16+3=56サイクル 22バイト

【書類名】　要約書

【要約】

【課題】　コンパイラによる最適化をユーザが緻密に制御することが可能な柔軟性の高いコンパイラを提供する。

【解決手段】　コンパイラ１００に対するユーザからの指示（オプション及びプラグマ）を検出する解析部１１０と、解析部１１０からの指示等に従って、ユーザによるオプション及びプラグマによって指定された個別的な最適化処理を行う処理部（グローバル領域割り付け部１２１、ソフトウェアパイプライニング部１２２、ループアンローリング部１２３、ｉｆ変換部１２４及びペア命令生成部１２５）からなる最適化部１２０等を備え、グローバル領域割り付け部１２１は、グローバル領域に配置する変数の最大データサイズの指定、グローバル領域に配置させる変数の指定、及び、グローバル領域に配置させない変数の指定に関するオプション及びプラグマに従った最適化処理を行う。

【選択図】　図１

出証特２００３－３００９８２１

認定・付加情報

特許出願の番号	特願２００２－１９５３０５
受付番号	５０２００９７７７１６
書類名	特許願
担当官	第七担当上席　　００９６
作成日	平成１４年　７月　４日

＜認定情報・付加情報＞

【提出日】　平成14年 7月 3日

出願人履歴情報

識別番号　〔０００００５８２１〕

１．変更年月日　１９９０年　８月２８日
［変更理由］　新規登録
住　所　大阪府門真市大字門真１００６番地
氏　名　松下電器産業株式会社